Non-Functional Requirements

(goals, requirements, specifications)

Lawrence Chung

Department of Computer Science

The University of Texas at Dallas

Non-Functional Requirements

Practices and Recommendations:
A Brief Synopsis

Why
What
Some Classification Schemes
NFRs and RE Processes
Some Individual NFRs
With Rational Unified Process and UML
With Volere Requirements Specification Templates

Why Non-Functional Requirements (NFRs)?

- Consider a brochure from an automobile manufacturer:
  - When you buy our car, you can now drive to a store…

- Consider a brochure from a cellular phone manufacturer:
  - When you buy our cellular phone, you can now call your friend.
  - Well, …

Why NFRs?

- **With automobiles:**
  - The basic function is transportation from one location to another.
  - "With *premium luxury, outstanding safety features and superior off-pavement capability*, … continues to exceed the high expectations of its owners, … continue to *set the standard for premium luxury* in its segment."

- **With cellular phones:**
  - The basic function is communication with another party
  - "… enhancements enable the *best possible* operation of your mobile … in various conditions. … The earpiece fits in either ear allowing for *convenient* and *discreet* access to all basic call controls. ... To *maximize call security*, the headset also supports encryption of the wireless connection for compatible … models.

- **With home networking:**

  "… is the total home networking solution … linking variety of digital home appliances as one. It enables you to enjoy *convenient, pleasant, and comfortable* living environment *at any time and any place*.

- **With CASE tool software:**
  - The basic function is provision of some services
  - "… is a *powerful, easy-to- use* application definition platform used by business experts to *quickly* assemble functionally *rich* simulations of Web-based applications *in a matter of hours*. … Using the *easy to learn*, drag-and-drop paradigm …, business people can *quickly* lay out the page flow of simulations and create *high fidelity* pages that *precisely* mimic not only the look and feel of the final application, …"

NFRs:
IEEE definition

"non functional requirement – in software system engineering, a software requirement that describes *not what* the software will do, *but how* the software will do it, for example, software performance requirements, software external interface requirements, design constraints, and software quality attributes. Nonfunctional requirements are difficult to test; therefore, they are usually evaluated subjectively."

General Observations

"non functional requirement – generally informally stated, often contradictory, difficult to enforce during development and evaluate for the customer prior to delivery"

What are Non-Functional Requirements?

- -ilities: understandability, usability, modifiability, inter-operability, reliability, portability, maintainability, scalability, (re-)configurability, customizability, adaptability, variability, volatility, traceability, …
- -ities: security, simplicity, clarity, ubiquity, integrity, modularity, nomadicity, …
- -ness: user-friendliness, robustness, timeliness, responsiveness, correctness, completeness, conciseness, cohesiveness, …
- …and many other things: performance, efficiency, accuracy, precision, cost, development time, low coupling, …

NFRs:
Some classification schemes - 1

➢ [Roman, IEEE Computer 1985]

- Interface requirements: describe how the system is to interface with its environment, users and other systems. E.g., user interfaces and their qualities (e.g., user-friendliness)

- Performance requirements: describe performance constraints involving
  - time/space bounds, such as workloads, response time, throughput and available storage space. E.g., "system must handle 100 transactions/second"
  - reliability involving the availability of components and integrity of information maintained and supplied to the system. E.g., "system must have less than 1hr downtime/3 months"
  - security, such as permissible information flows
  - survivability, such as system endurance under file, natural catastrophies

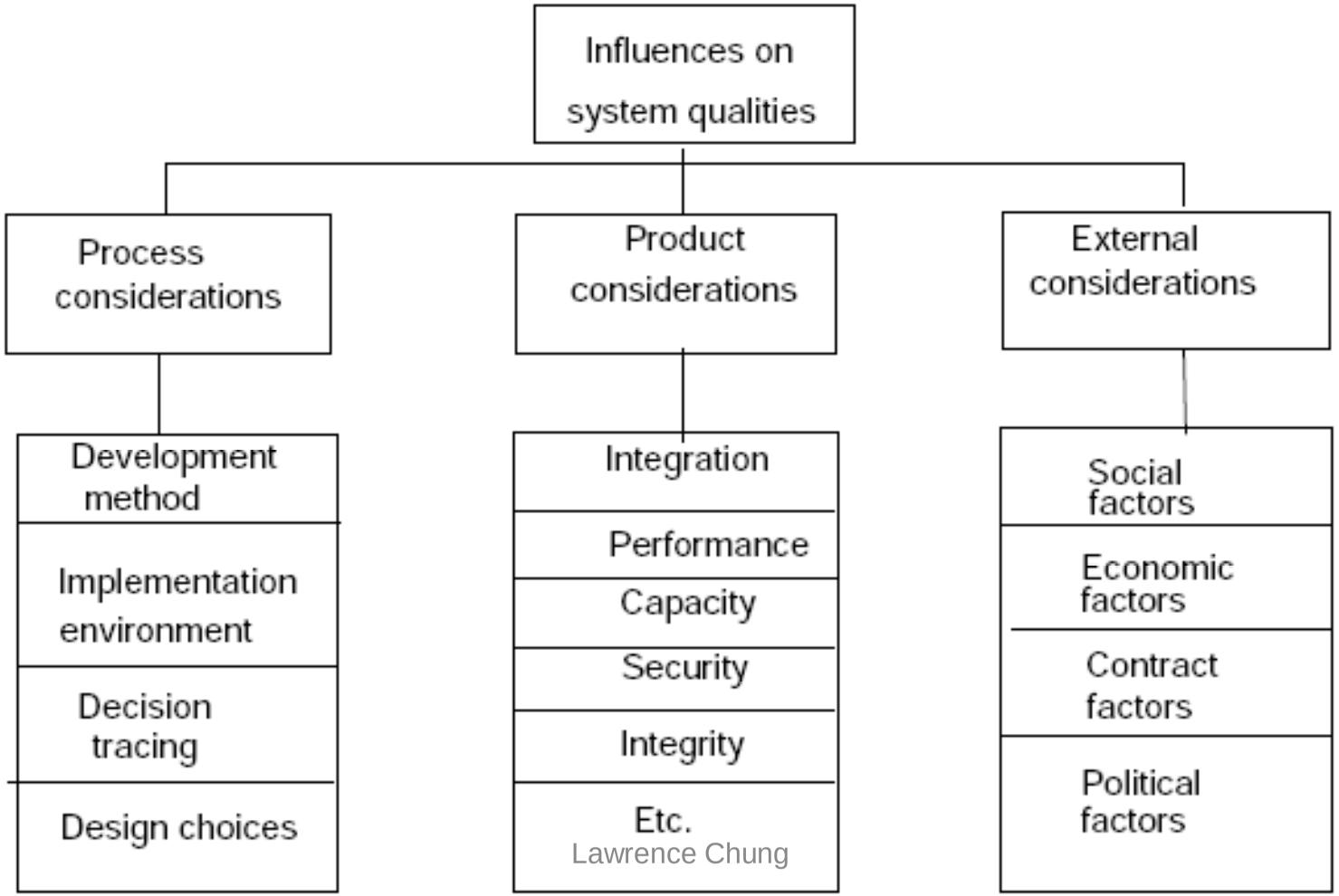
- Operating requirements: include physical constraints (size, weight), personnel availability, skill level considerations, system accessibility for maintenance, etc.

- Lifecycle requirements: can be classified under two subcategories:
  - quality of the design: measured in terms such as maintainability, enhanceability, portability.
  - limits on development, such as development time limitations, resource availability, methodological standards, etc.
- Economic requirements: immediate and/or long-term costs

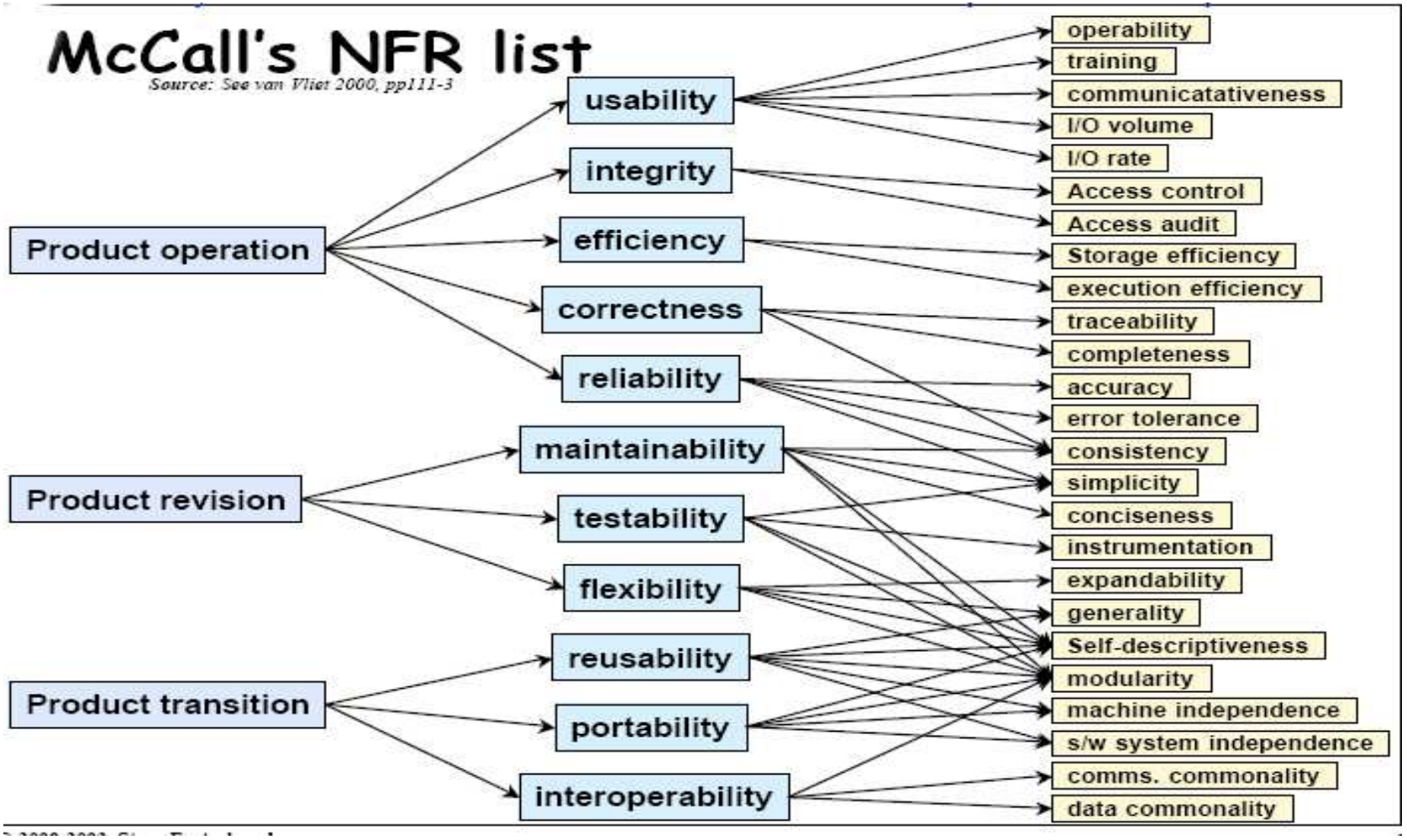
- Political requirements

NFRs:
Some classification schemes - 2

- Process, Product and External considerations [Sommerville 1992]



NFRs:
Some classification schemes - 3



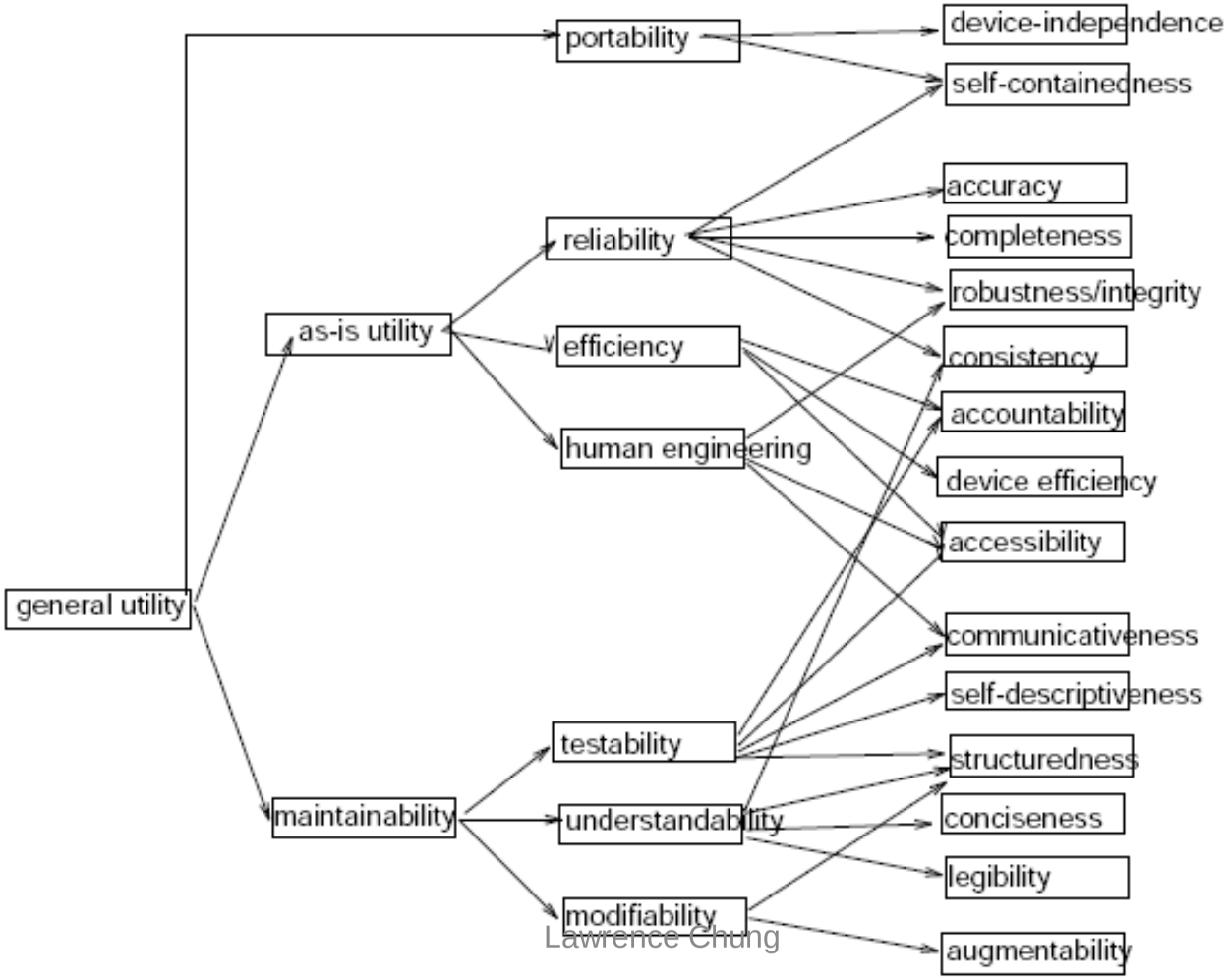
NFRs:
Some classification schemes - 4

➢ **Dimensions of Quality –Components of FURP+** [Grady1992]

Functionality	Feature set capabilities, security, generality
Usability	Human factors aesthetics, consistency, documentation
Reliability	Frequency/severity of failure, recoverability, predictability, accuracy, MTBF
Performance	Speed efficiency, resource usage, throughput, response time
Supportability	Testability, Adaptability, Compatibility, Serviceability, Localizability, Extensibility, Maintainability, Configurability, Installability, Robustness

NFRs:
Some classification schemes - 5

➢ Software Quality Tree [Boehm 1976]



NFRs & RE Processes: Why?

Quality of product ⇐ **Quality of Process**

- Garbage in garbage out, so get the right requirements
- Garbage thru garbage out, so get the right process

Evolution is inevitable – traceability is a virtue

Approaches to NFRs

Measurement of products or systems is absolutely fundamental to the engineering process. I am convinced that measurement as practised in other engineering disciplines is *IMPOSSIBLE* for software engineering [Sommerville; http://www.utdallas.edu/~chung/SE3354Honors/IEEInaugural.pdf]

- **Product vs. Process?**

The most important things can't be measured [Deming]

  - Product-oriented Approaches
    - Focus on system (or software) quality
    - Aim is to have a way of measuring the product once it's built – metrics
  - Process-oriented Approaches
    - Focus on how NFRs can be used in the design process
    - Aim is to have a way of making appropriate design decisions

- **Quantitative vs. Qualitative?**
  - Quantitative Approaches
    - Find measurable scales for the quality attributes
    - Calculate degree to which a design meets the quality targets
  - Qualitative Approaches
    - Study various relationships between quality goals
    - Reason about trade-offs etc.

Not everything that can be counted counts, and not everything that counts can be counted. [Albert Einstein]

NFRs & RE Processes:

So, where are NFRs in an RE Process?

- ***Before FRs?***
- ***After FRs?***
- ***At the same time with FRs?***
- ***…and what about Business objectives/goals, system architectures, system models, SS, SRS, …?***

But, should we perhaps better know about the various relationships between NFRS and such and such,

before answering these questions,

more clearly,

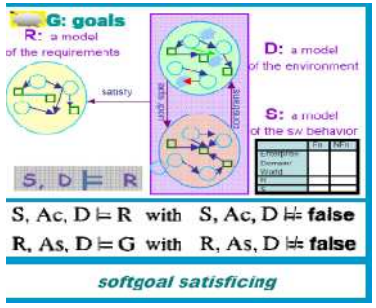
understandably,

concisely,

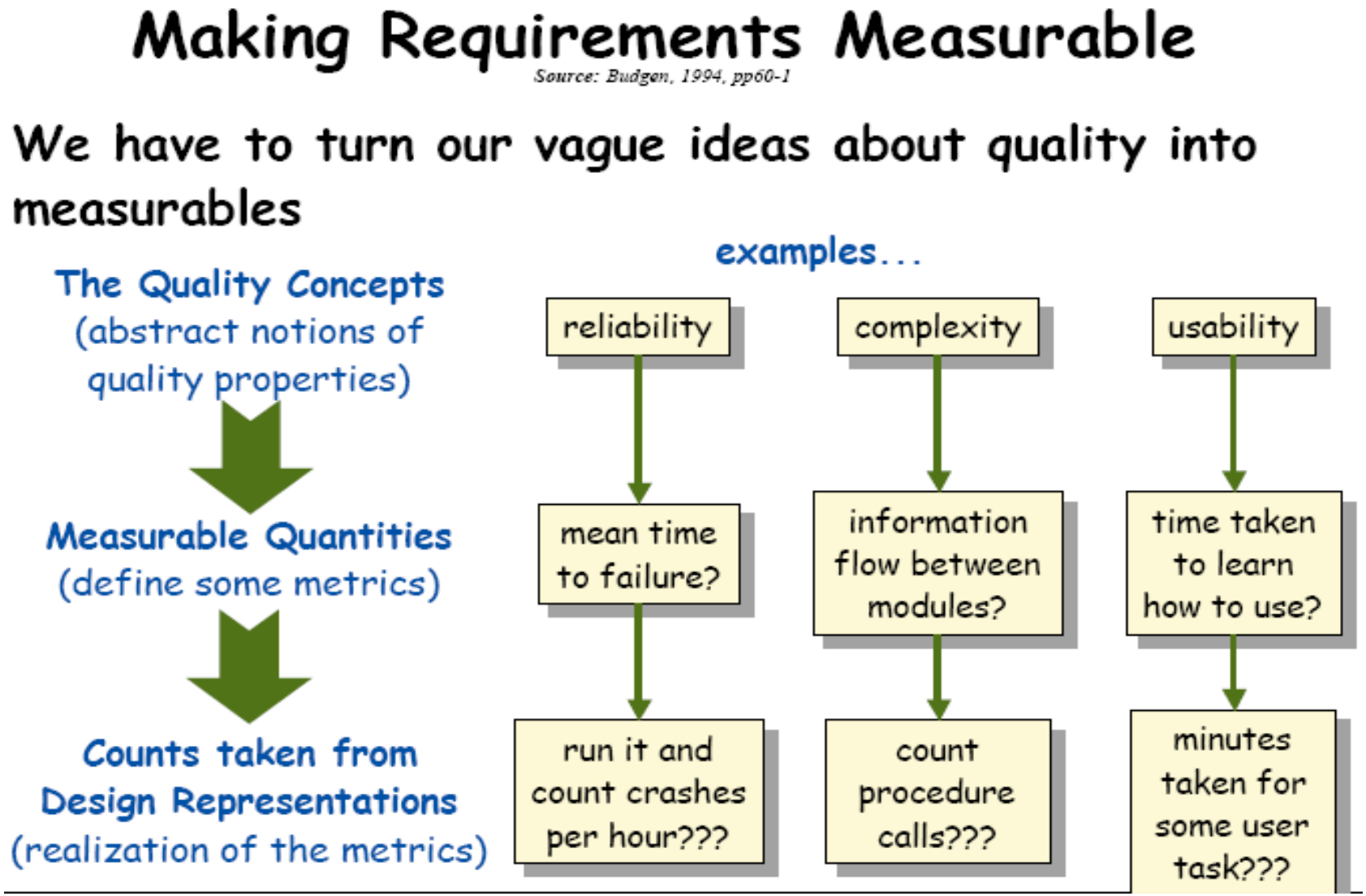
precisely,

agreeably, …?

M, Prog |= S; G^s, S, D |= R; (G^s, R, D |= G) V (G^s, R, D |~ G); (G |= ¬P) V (G |~ ¬P)



Product-oriented approaches



Product-oriented approaches

^ Quality Metrics:

Property	Metric
Speed	transactions/sec, response time, screen refresh time
Size	KBytes, LOCs, Function Points, Complexity measures
Ease of use	transactions/sec, response time, screen refresh time

* usual metrification process:
  1. *determine a set of desirable attributes (i.e., ilities)*
  2. *determine relative importance/weight of such attributes*
  3. *evaluate the quality (rating) of each of the attributes*
  4. *compute weighted rating for each*
  5. *sum up all the weighted ratings*

Property	relative weight	rating	weighted rating
Speed	.3	6	1.8
Size	.6	5	3.0
Ease of use	.1	7	0.7
Overall Quality			5.5/10

* an inexact science at this point
* however, aids in understanding the factors that affect sw quality
  - a first-cut approximation
  - very poor quality factor

NFRs: Portability

- The degree to which software running on one platform can easily be converted to run on another platform
- E.g., number of target statements (e.g., from Unix to Windows)
- Hard to quantify, since it is hard to predict what a “next generation” platform might be like
- Can be enhanced by using languages, OSs and tools that are universally available and standardized.

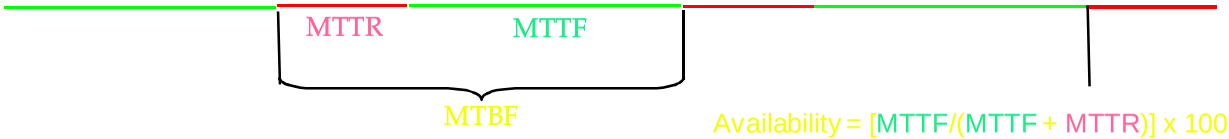
  E.g., C/C++/C#/Java

  J2EE/J2ME/.NET

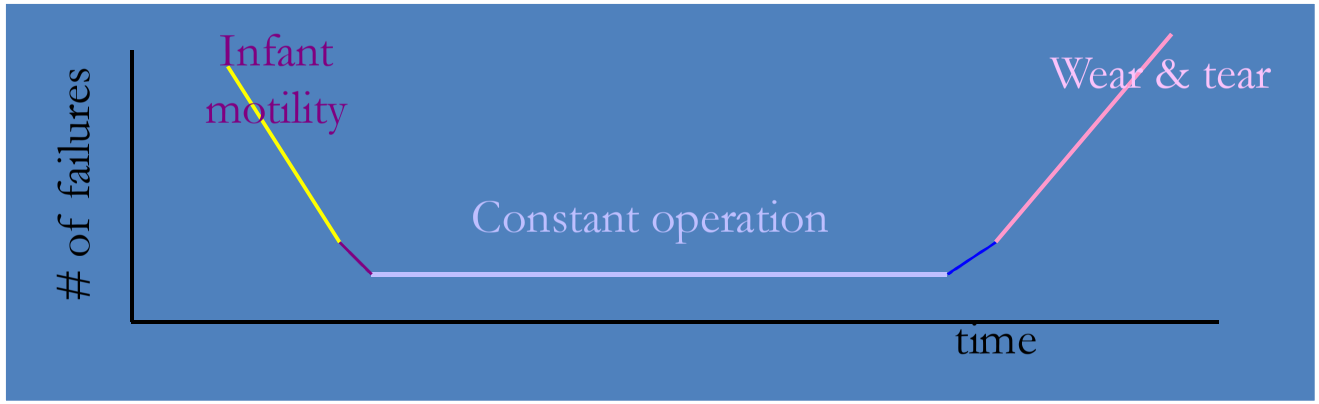
NFRs: Reliability

- **the ability of the system to behave consistently in a user-acceptable manner when operating within the environment for which the system was intended.**

- **theory and practice of hardware reliability are** well established; some try to adopt them for software



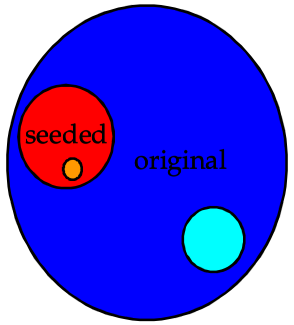
- one popular metric for hardware reliability is mean-time-to-failure (MTTF)
  "Bathtub" curve characterizes MTTF:



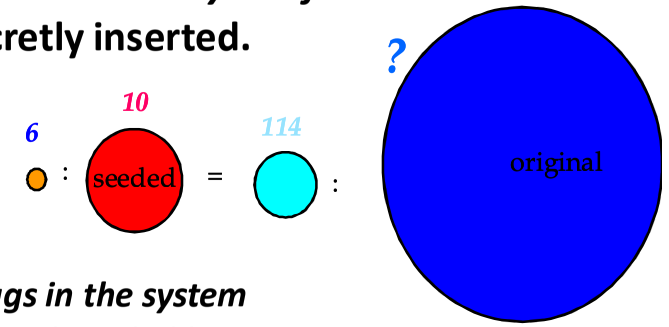
- **Infant mortality:**
  Given a large population of a particular component, many will fail soon after development due to inaccuracies in the manufacturing process;
- **Issues:**
  Do 2 different software copies have different characteristics?
  Does software wear & tear by decomposition?
  Does software obey physical laws?

NFRs: Reliability

- **Sometimes reliability requirements take the form:**
  "The software shall have no more than X bugs/1K LOC"
  But how do we measure bugs at delivery time?



- ***Bebugging*** ***Process - based on a Monte Carlo technique for statistical analysis of random events.***
  1. before testing, a known number of bugs (seeded bugs) are secretly inserted.
  2. estimate the number of bugs in the system
  3. remove (both known and new) bugs.



# of detected seeded bugs/ # of seeded bugs = # of detected bugs/ # of bugs in the system
# of bugs in the system = # of seeded bugs x # of detected bugs /# of detected seeded bugs

Example: ***secretely seed 10 bugs (say, in 100 KLOC)***
an independent test team detects 120 bugs (6 for the seeded)
# of bugs in the system = 10 x 120/6 = 200
# of bugs in the system after removal = 200 - 120 - 4 = 76

190 – 114; 100000 – (190-114)/100000

- ***But*****, deadly bugs vs. insignificant ones; not all bugs are equally detectable; (Suggestion [Musa87]:**

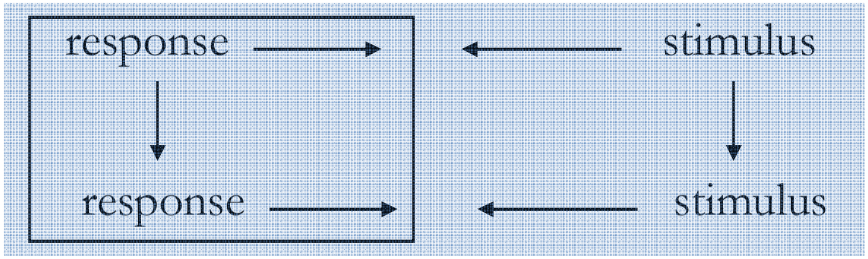
NFRs: Efficiency

- refers to the level at which a software system uses scarce computational resources, such as CPU cycles, memory, disk space, buffers and communication channels

- can be characterized along a number of dimensions:
  Capacity: maximum number of users/terminals/transactions ...

  Degradation of service: what happens when a system with capacity X widgets per time unit receives X+1 widgets?
  - Let the system handle the load, perhaps with degraded performance
  - Let the system crash

  Timing constraints: Let stimulus refer to an action performed by the user/environment, and response refer to an action generated by the system.



- **stimulus-response:** e.g., "the system will generate a dial tone within 10 secs from the time the phone is picked up"
- **response-response:** e.g., "the system will record that the phone is in use no later than 1 micro-second after it had generated a dial tone"
- **stimulus-stimulus:** e.g., "the user will type her password within 15 secs from typing her login name"
- **response-stimulus:** e.g., "the user will start dialing the phone number within 1 minute from getting the dial tone"

NFRs:
Usability

- broadly – quality; fit to use
  narrowly - good UI

- **Usability inspection**:
  finding usability problems in UI design, making recommendations for fixing them, and improving UI design.

- ***Heuristics:*** a set of criteria against which usability of UI design is evaluated

- ***"9 usability heuristics"*** [Nielsen90]
  - *Promptness* no undue delay in accepting info items and responding to requests
  - *Tolerance* no hang-ups against errors, delays, unexpected behavior, etc.
  - *Guidance* providing guidance for correcting errors, generating reminders, etc.
  - *Coherence* ... ...

- ***"10 usability heuristics"*** [Molich and Nielsen90]
  - *Simple and natural dialogue; Speak the user's language*
  - *Minimize the user's memory; Consistency; Feedback*
  - *Clearly makred exits; Shortcuts*
  - *Precise and constructive error messages; Prevent errors*
  - *Help and documentation*

NFRs:
Usability

- All users will be satisfied with the usability of the product.
- 95% of all users will be satisfied with the usability of the product.

- 95% of the users will be able to complete representative tasks without requiring assistance (e.g., modifying exclusion date set)
- 95% of the users will be able to complete representative tasks by the third attempt without requiring assistance

- 95% of the users will be able to complete tasks X Y Z by the third attempt without requiring assistance

- 95% of the users will be able to complete tasks X Y Z in less than 10 minutes without requiring assistance
- 95% of the users will be able to complete task X in less than 10 minutes without requiring assistance
- 80% of the users will be able to complete task Y in less than 10 minutes
- 77% of the users will be able to complete task Z in less than 5 minutes

Dependability

- **Dimensions of Dependability**
  - **Availability** - The ability of the system to deliver services when requested
  - **Reliability** - The ability of the system to deliver services as specified
  - **Safety** - The ability of the system to operate without catastrophic failure
  - **Security** - The ability of the system to protect itself against accidental or deliberate intrusion

- **Cost of development** - Geometric rise in cost from low dependability to highest
- **Effects of low dependability**
  - Often unused
  - Failure recovery costs may be high
  - Difficult to retrofit dependability
  - Loss of information
- **Repeatable improvement process** helps
  - CMM -SEI
  - More later

- Critical Systems
  - Safety critical
  - Mission critical
  - Business critical
- Dependability a key aspect
  - A system failure causes
    - Significant economic loss
    - Physical damage
    - Threat to or loss of human life

Dependability

- Cost of failure
  - direct
    - Loss of life / Injury
    - Loss of business
  - Indirect
    - Litigation
    - Good will
- **Availability** and **Reliability**
  - Factors effecting
    - Environment office versus university
    - Perception (frequency of occurrence)
- Degrees
  - Failure - service that is expected is not delivered
  - Error – behavior that does not conform to the specification
  - Fault – incorrect state – un-anticipated
  - Human error

- Improve **reliability**
  - Fault avoidance
  - Fault detection and removal – testing and debugging
  - Fault tolerance - self checking and redundancy
- Errors of this type are random
  - Remain after testing due to unforeseen combinations of input or use
  - Random based on user methods
    - Not all inputs done the same
    - Learn to avoid
    - Therefore removal of some faults will not improve perception

Dependability - Safety

- Ability to operate normally or abnormally without threat to life or environment
- Classes
  - Primary safety critical
    - Embedded as controller
  - Secondary
    - There output could effect indirectly other processes (CAD)
- Reasons for less than 100% certainty of fault tolerant/free
  - Incomplete specification
  - Hardware malfunction – causing exceeded limits in software
  - Incorrect input

- Methods to lessen chance of safety failure
  - Hazard avoidance
    - Added control features (I.e. two man rule)
  - Hazard detection and removal
    - Scans for known causes and cause preventive action
  - Damage limitation (control)
    - Firewalls and other protective reactions to results
- Terms
  - Accident
  - Hazard
  - Damage
  - Hazard Severity
  - Hazard Probability
  - Risk

Specification

- **Safety**
  - **IEC 61508 safety life cycle**
    - **Concept to death**
      - **Hazard analysis**
      - **Safety requirements definition**
      - **Planning , validation, development, external risk reduction**
      - **Separate safety validation – installation and commissioning**
      - **O&M**
      - **Decommissioning**
  - **Hazard and Risk Analysis**
    - **Iterative process**
      - **Hazard Identification**
        - » **Hazard description**
      - **Risk analysis and hazard classification**
        - » **Risk assessment**
      - **Hazard decomposition**
        - » **Analysis as to potential causes (fault-tree analysis)**
      - **Risk reduction analysis**
      - **Preliminary safety requirements**

- Fault tree
  - Deductive – start with a hazard
  - Inductive – start with failure
  - Fault tree starts with the failure and works backwards to potential causes
- Risk assessment
  - Classifications
    - Intolerable
    - As low as reasonably practical (ALARP)
    - Acceptable
  - For each hazard
    - Probability
    - Severity
    - Estimated risk
- Risk reduction
  - Avoidance
  - Detection and removal
  - Damage limitation

Dependability - Security

- Lack of **security** comprise to **availability** and **reliability**
- Types
  - Denial of service
  - Corruption of programs or data
  - Unauthorized disclosure
- Terms
  - Exposure
  - Vulnerability
  - Attack
  - Threats
  - Controls
- Methods
  - Vulnerability avoidance
  - Detection and neutralization
  - Damage limitation

- **Security Specification**
  - Similar to safety
  - Impractical to specify
  - Usually are "shall not"
- Cycle in General
  - Asset ID and evaluation
    - Degree of importance
  - Threat analysis and risk assessment
  - Threat assignment lists all threats against each asset
  - Technology analysis what is available to counteract
  - Security specification

Specification

- **Requirements specification**
  - Functional for error detection and recovery
  - Non functional for **reliability** and **availability**
  - Shall not requirements
- **Reliability** specification
  - Hardware
  - Software
  - Operator
- Decrease probability of failure
  - For a series of dependent components Pt = sum of P1 to Pn
  - But if there are n replicated (redundant) and independent components then the Pt=pa to the nth
- Metrics for **reliability**
  - POFD probability of failure on demand .0001 = 1 on 10000
    - Systems with unpredictable demand over long time periods – emergency systems
  - ROCOF Rate of failure occurrence 2/1000
    - Systems with a regular demand atm/airline reservations
  - MTTF Mean time to Failure avg time between observed failures 500 = avg of 1 in 500 time units
    - Systems with long transactions (auto save)
  - AVAIL probability system is available at any given time .999 equals in every given 1000 time units system is likely to be available for 999 of these
    - Systems of continuous service; tp switch

- Non-functional **reliability** requirements
  - ID type of failure to occur
  - Partition them into
    - Transient
    - Permanent
    - Recoverable
    - Unrecoverable
    - Non-corrupting
    - Corrupting
  - Define the appropriate requirement (metric)
    - E.g. recoverable w/intervention – POFOD
    - If automatic the ROCOF
  - Assign a proper metric as a functional reliability metric

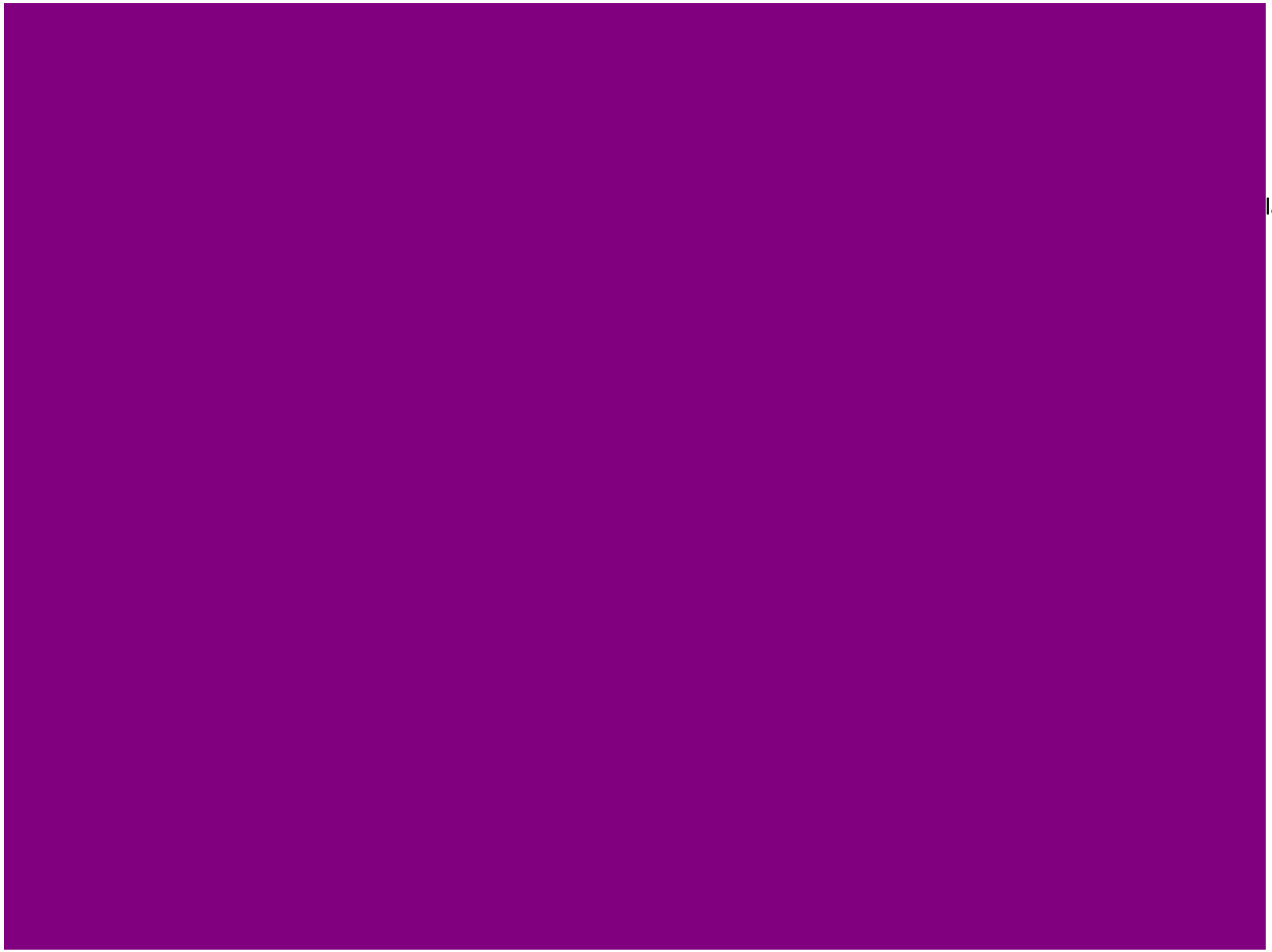
NFRs:
With Rational Unified Process and UML

Home Appliance Control System
Vision
Version 1.2

Revision History

Date	Version	Description	Author

NFRs:
With Rational Unified Process and UML



lance.

NFRs: With Volere Requirements Specification Template

The Atlantic Systems Guild Limited

NFRs: With Volere Requirements Specification Template

10 Look and Feel Requirements

10a. The interface

Content

The section contains requirements relating to spirit of the interface. Your client may have given you particular demands such as corporate branding, style, colors to be used, degree of interaction and so on. This section captures the requirements for the interface rather than the design for the interface.

Motivation

To ensure that the appearance of the product conforms to the organizationÕs expectations.

Examples

The product shall comply with corporate branding standards.

The product shall be attractive to a teenage audience.

The product shall appear authoritative.

Considerations

Interface design may overlap the requirements gathering process. This particularly true if you are using prototyping as part of your requirements process. As prototypes develop it is important to capture the requirements that relate to the look and feel. In other words, be sure that you understand your client's intentions for the product's look and feel. Record these as requirements instead of merely having a prototype to which the client has nodded his approval.

10b. The style of the product

Content

A description of salient features of the product that are related to the way a potential customer will see the product. For example, if your client wants the product to appeal to the business executive, then a look and feel requirement is that the product has a conservative and professional appearance. Similarly if the product is for sale to children, then the look and feel requirement is that it be colorful and look like it's intended for children. …

Motivation

Given the state of today's market and people's expectations, … Once the functional requirements are satisfied, it is often the appearance of products that determines whether they are successful or not. …

Considerations

The look and feel requirements specify the your client's vision of the product's appearance. The requirements may at first seem

NFRs: With Volere Requirements Specification Template

11 Usability and Humanity Requirements

11a. Ease of use.

Content

This section describes your client's aspirations for how easy it will be for the intended users of the product to operate it. The product's usability is derived from the abilities of the expected users of the product and the complexity of its functionality.

The usability requirements should cover such things as:

Efficiency of use - how quickly or accurately the user can use the product.

Ease of remembering - how much is the casual user expected to remember about using the product

Error rates - for some products it is crucial that the user commits very few, or no, errors.

Overall satisfaction in using the product - this is especially important for commercial, interactive products where there is a lot of competition. Web sites are good example of this.

Feedback - how much feedback does the user need in order to feel confident that the product is actually accurately doing what the user expects. The necessary degree of feedback will be higher for some products (eg: safety critical) than in others.

Motivation

To guide the product's designers into building a product that will meet the expectations of its eventual users.

Examples

The product shall be easy for 11 year-old children to use.

The product shall help the user to avoid making mistakes.

The product shall make the users want to use it.

The product shall be used by people with no training, and possibly no understanding of English.

Fit Criterion

These examples may seem simplistic, but they do express the intention of the client. To completely specify what is meant by the requirement it is necessary to add a measurement of acceptance. We call this a fit criterion. The fit criterion for the above examples would be:

[An agreed percentage, say 90%] of a test panel of 11 year olds shall be able to successfully complete [list of tasks] within [specified time]

One month's use of the product shall result in a total error rate of less than [an agreed percentage, say 2%]

An anonymous survey shall show that [an agreed percentage, say 75%] of the users are regularly using the product after [an agreed time] familiarization period.

Considerations

Non-Functional Requirements

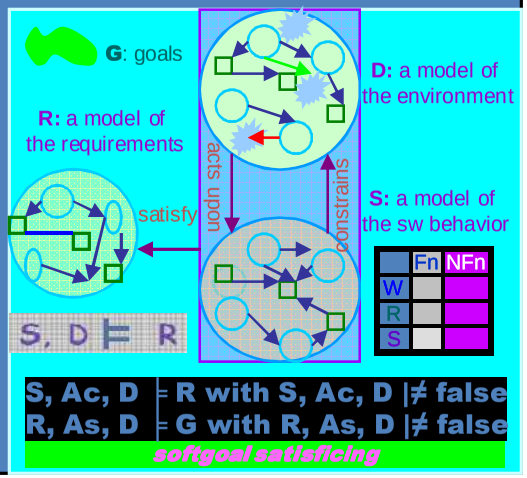
Practices and Recommendations:

A Brief Synopsis

- Why
- What
- Some Classification Schemes
- NFRs and RE Processes
- Some Individual NFRs
- With Rational Unified Process and UML
- With Volere Requirements Specification Templates

Non-Functional Requirements

What - Essential Concepts



$M, Prog \models S; G^s, S, D \models R; (G^s, R, D \models G) \vee (G^s, R, D \mid\sim G); (G \models \neg P) \vee (G \mid\sim \neg P)$

NFRs:
functional vs. non-functional: a mathematical perspective

- (mathematical) function:

  f_1: I -> O
  f_2: I_1 X I_2 -> O
  e.g.: sum: R X R -> R

- non-functional:
  - How fast can it be done?
  - How precise is the answer?
  - How easy is it to figure out how to use it?
  - How robust is it concerning the 2^{nd} input of f_2?
  - Who can use it?
  - Can it be changed easily?
  - How much would it cost to design and implement it?

NFRs:
functional vs. non-functional: a mathematical perspective

- (mathematical) function:

$$f(x, y) = f_1(f_2(x), f_3(y))$$

- non-functional:

$$\boldsymbol{n}f(x, y) = \boldsymbol{n}f_1(\boldsymbol{n}f_2(x), \boldsymbol{n}f_3(y))$$

$$\boldsymbol{n}f(x, y) = \boldsymbol{n}f_1(\boldsymbol{n}f_2(\boldsymbol{n}(x)), \boldsymbol{n}f_3(\boldsymbol{n}(y)))$$

NFRs:
subjective, graded, interacting

- Subjective vs. objective:

  subjective ⟷ objective

- Graded:

  worse ⟷ better

  expensive ⟷ cheaper

  slower ⟷ faster

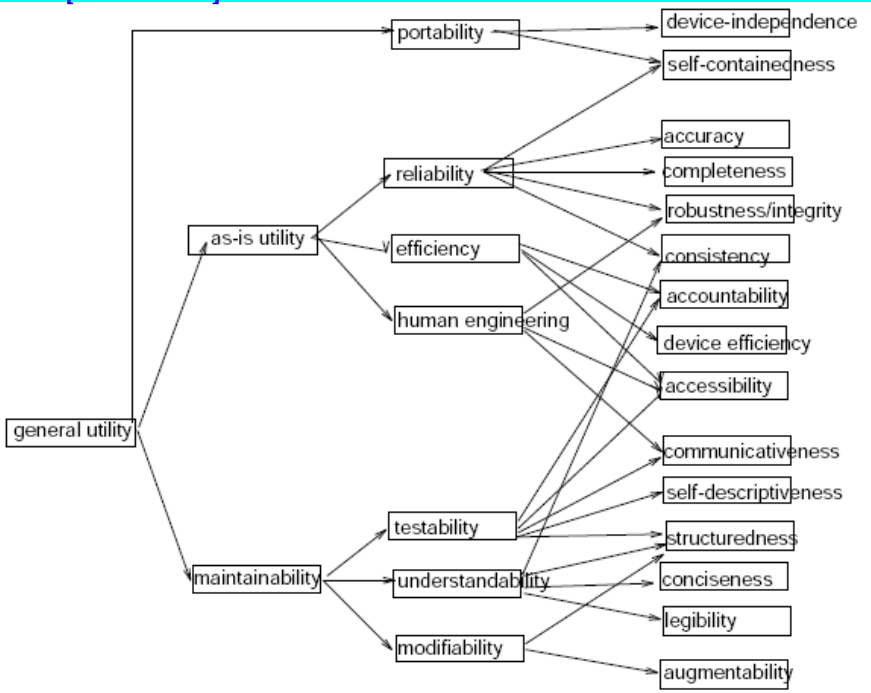
- Interacting:
  - Conflicting: the whole is less than the sum of its parts
  - Synergistic: the whole is more than the sum of its parts

NFRs:
subjective in both definitions & solutions

Classification 1 [Roman, IEEE Computer 1985]

- **Interface requirements**: describe how the system is to interface with its environment, users and other systems. E.g., user interfaces and their qualities (e.g., user-friendliness)
- **Performance requirements**: describe performance constraints involving
  - time/space bounds, such as workloads, response time, throughput and available storage space. E.g., "system must handle 100 transactions/second"
  - reliability involving the availability of components and in
    system. E.g., "system must have less than 1hr downtime/
  - security, such as permissible information flows
  - survivability, such as system endurance under file, natural
- **Operating requirements**: include physical constraints (si
  considerations, system accessibility for maintenance, etc
- **Lifecycle requirements**: can be classified under two sub
  - quality of the design: measured in terms such as maintain
  - limits on development, such as development time limitati
    etc.
- **Economic requirements**: immediate and/or long-term
- **Political requirements**

Classification 5 - Software Quality Tree [Boehm 1976]



NFRs:
subjective in both definitions & solutions

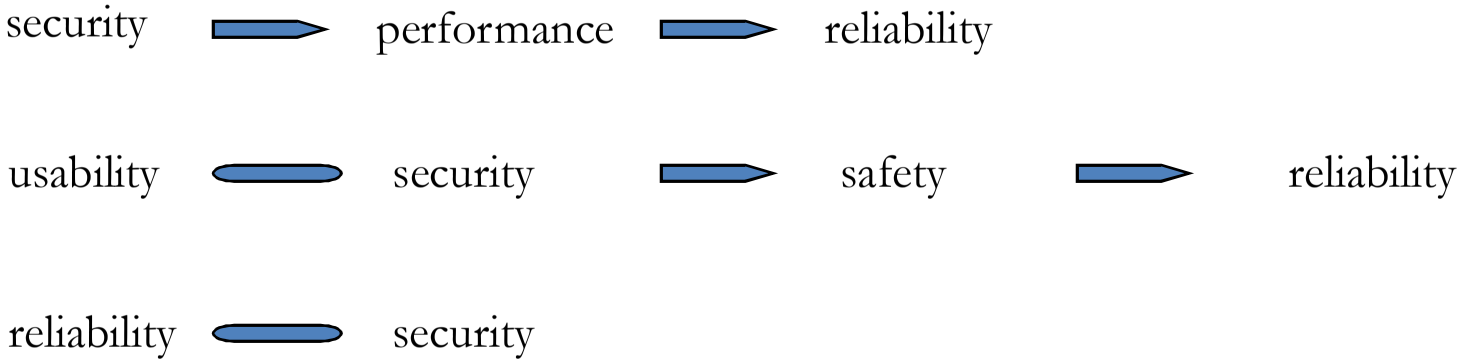
Consider "security" – problem is subjective

- Protection of data alone, fine with Chris
- Protection of data, and data availability, fine with Pat
- Protection of data, and data availability, and data accuracy, fine with Alex
- Protection of data, and data availability, and data accuracy, and filtering of viruses, fine with Neo
- Protection of data, and data availability, and data accuracy, and filtering of viruses, and blocking adware, fine with Gail

Consider "security" – solutions are subjective

- A password authentication fine with Chris
- A password authentication, with periodic change, fine with Pat
- A password, together with a fingerprint verification, fine with Alex
- A password, with a fingerprint verification rechecked every hour, fine with Neo
- A password, with a fingerprint verification rechecked every hour, and co-presence of two people, fine with Gail

NFRs:
subjective – and also relative in priorities



NFRs:
graded in both definitions and solutions – and relative

- Protection of data alone good
- A password authentication alone bad

- Protection of data alone << Protection of data, and data availability
- A password authentication << A password, together with a fingerprint verification

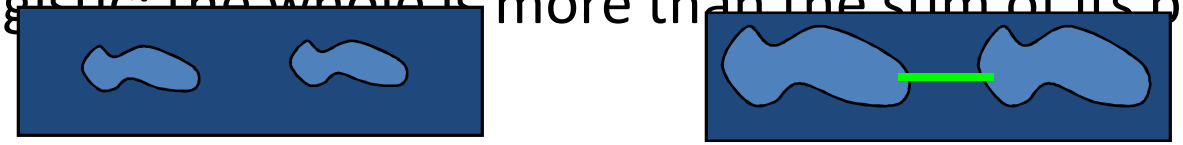
NFRs: interacting

– Conflicting: the whole is less than the sum of its parts

✓ A password, with a fingerprint verification rechecked every hour, fine for security

✓ Simplicity is the key for ease-of-use

– Synergistic: the whole is more than the sum of its parts



✓ A password, with a fingerprint verification rechecked every hour, fine for security

✓ Restricted access is good for data accuracy

Non-Functional Requirements

What - Essential Concepts

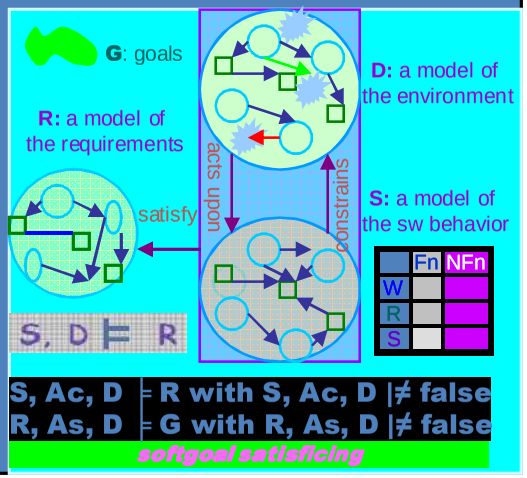
- ❑ non-functional,
- ❑ subjective,
- ❑ graded,
- ❑ interacting
- ❑ – and relative
- ❑ - in both definitions & solutions

Non-Functional Requirements

How 1 - Essential Tasks

Goal-oriented analysis focuses on the description and evaluation of alternatives and their relationship to the organizational objectives.

$M, Prog \models S; G^s, S, D \models R; (G^s, R, D \models G) \vee (G^s, R, D \mid\sim G); (G \models \neg P) \vee (G \mid\sim \neg P)$



NFRs:
functional vs. non-functional: a mathematical perspective

- (mathematical) function:
  f_1: I -> O
  f_2: I_1 X I_2 -> O
  e.g.: sum: R X R -> R

  $f(x, y) = f_1(f_2(x), f_3(y))$
  $nf(x, y) = nf_1(nf_2(x), nf_3(y))$
  $nf(x, y) = nf1(nf2(n(x)), nf3(n(y)))$

- non-functional:
  - How fast can it be done? ***Fast, Fast(f), Fast(f_2)***
  - How precise is the answer? ***Precise, Precise(f), Precise(O)***
  - How easy is it to figure out how to use it?
    Easy-to-learn, Easy-to-learn(f), Easy-to-learn(f_2), Easy-to-learn(x)
  - How robust is the input? ***Robust, Robust(I_1) , Robust(I_2)***
  - Who can use it? ***Security, Security(f), Security(I), Security(O), Security (f_2), Accessibility, Accessibility(f), Accessibility(O)***
  - Can it be changed easily?
    Changeability, Changeability(f), Changeability(f_2)
  - How much would it cost?
    Cost, Design-cost(f), Implementation-cost(f), Testing-cost(f_2)

The NFR Framework

{Chung et.al.} Non-functional Requirements in Software Engineering [Software Engineering lecture notes]

- Based on traditional framework for problem solving in AI [Nilsson]
  - Establish the goals
  - Introduce sub-goals to satisfy the goal where the relationship is AND or OR
    - AND goal is satisfied when all of sub goals are satisfied
    - OR goal is satisfied when any of the sub goals are met
  - Continue until you cannot decompose further

- ***Softgoal***: no clear-cut definition and or criteria as to whether it is satisfied or not , since NFRs are subjective, relative, and interdependent
  - Introduce concept of ***satisficing***
  - Provide basis for saying the softgoal can ***contribute positively or negatively, fully or partially***, to some degree in satisfying other softgoals (i.e., achieved not absolutely but within acceptable limits).

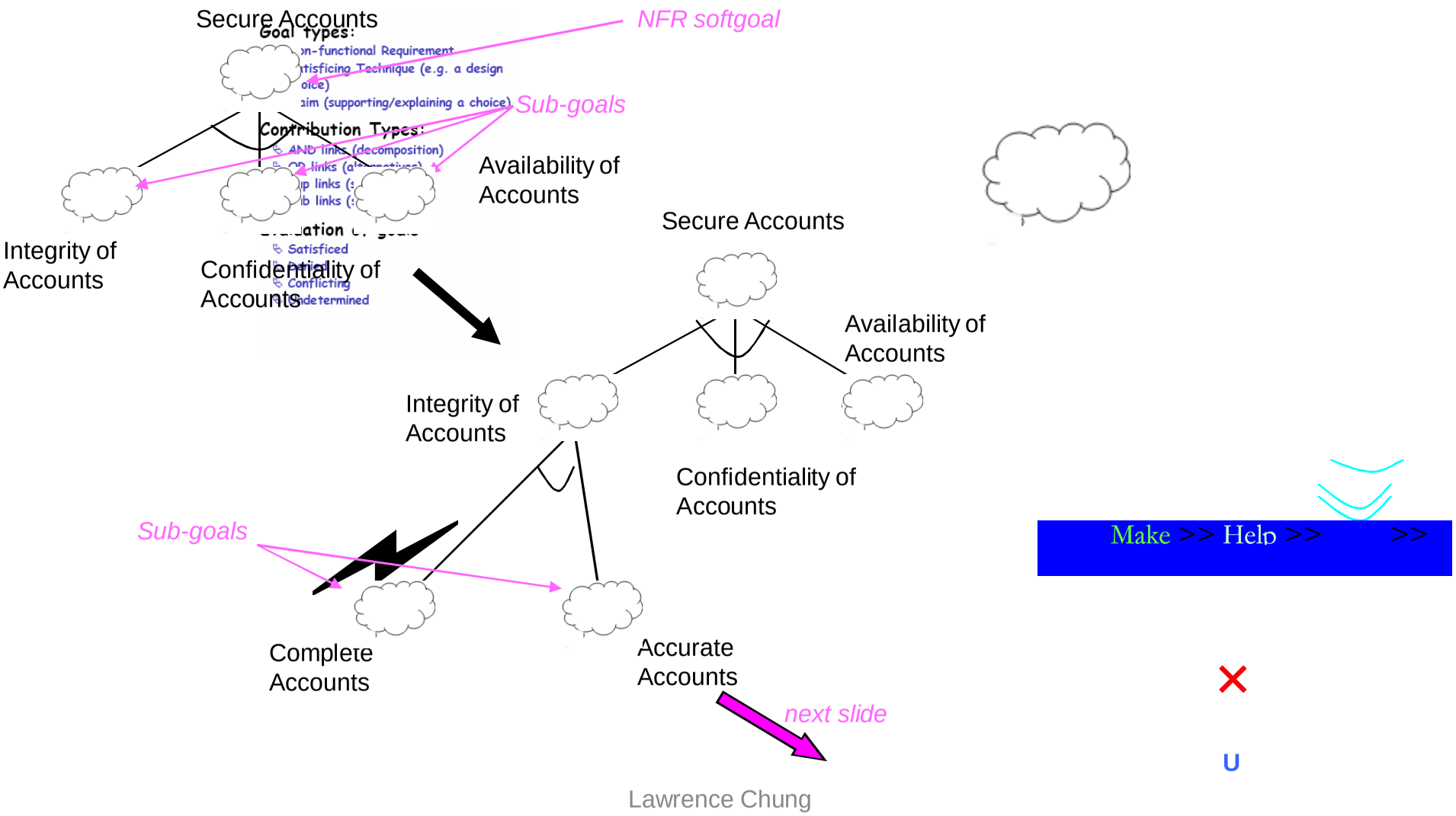
- ***Softgoal Interdependency Graphs*** (SIGs)
  - For modeling non-functional requirements and interdependencies between them

- Introduces ***Catalogues*** of NFRs much like patterns for design are built

Qualitative in nature, Process oriented

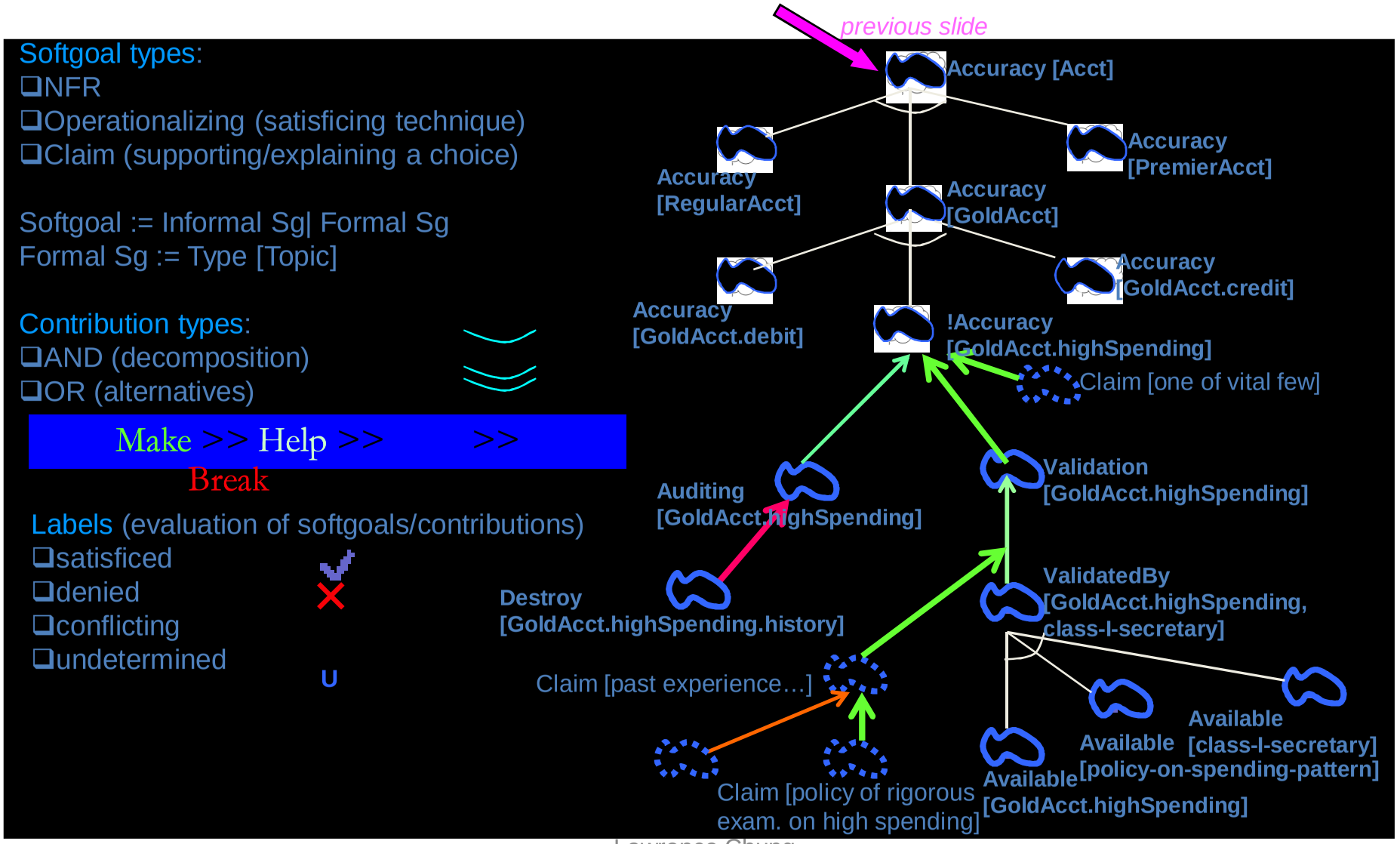
The NFR Framework

Qualitative in nature, Process oriented



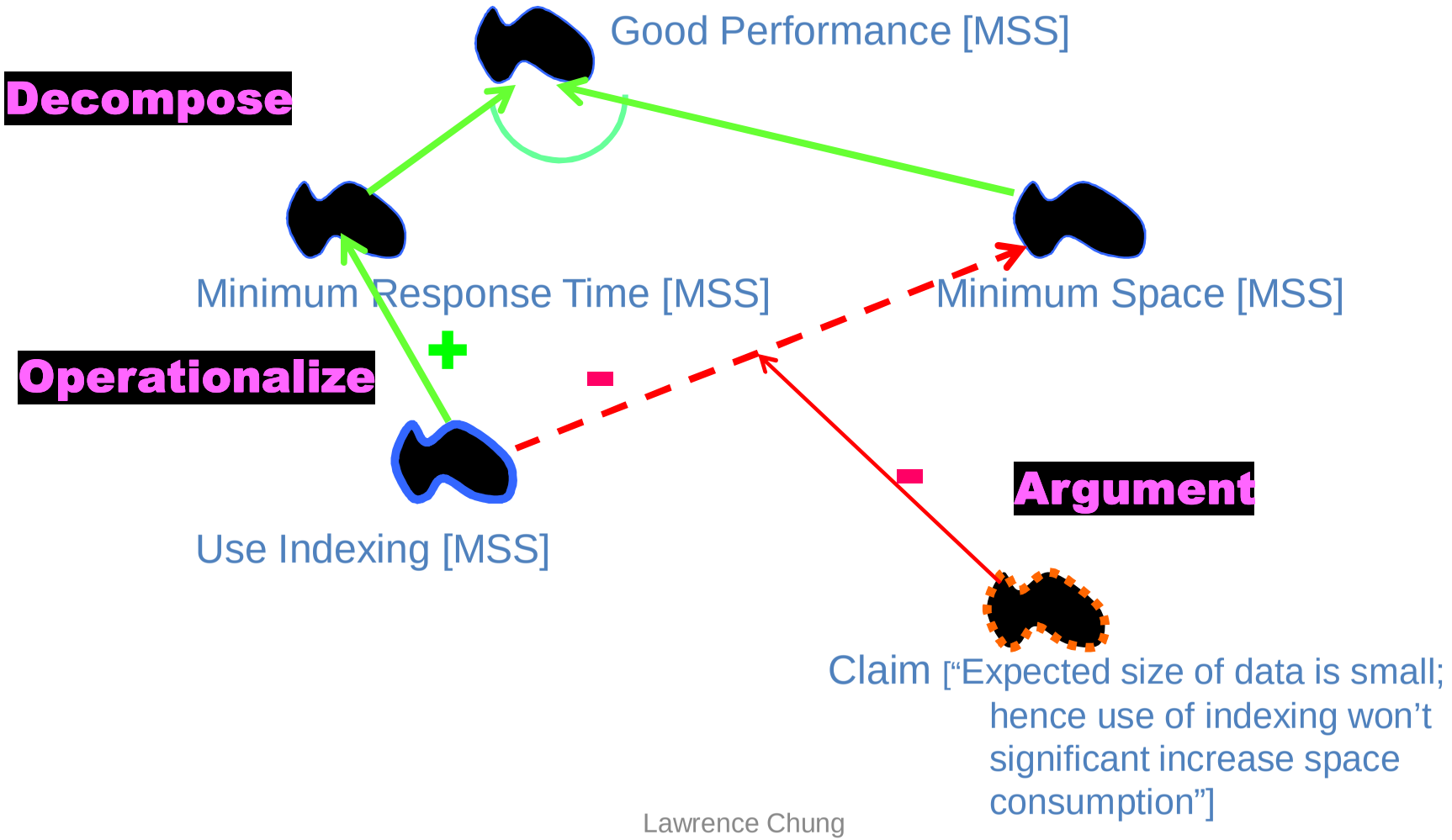
The NFR Framework

Softgoal Interdependency Graph (SIG)



The NFR Framework

Softgoal Interdependency Graph (SIG):
Three types of refinements

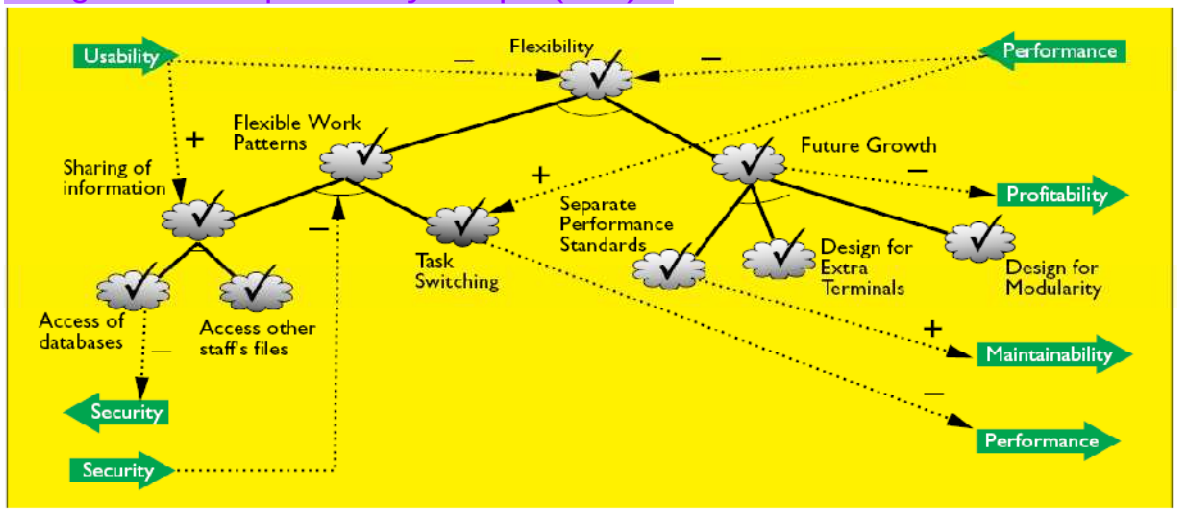


The NFR Framework

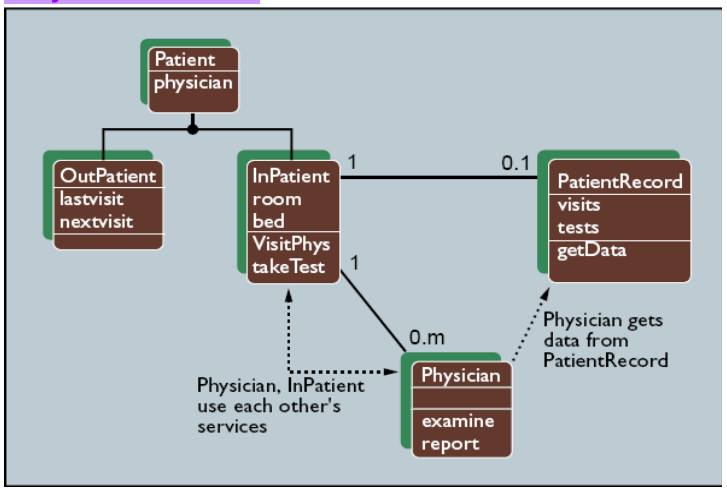
[J. Mylopoulos, L. Chung, E. Yu, "From object-oriented to goal-oriented requirements analysis ", *CACM*, pp31-37.ACM Press]

Example: A small portion of a hospital model for requirements analysis

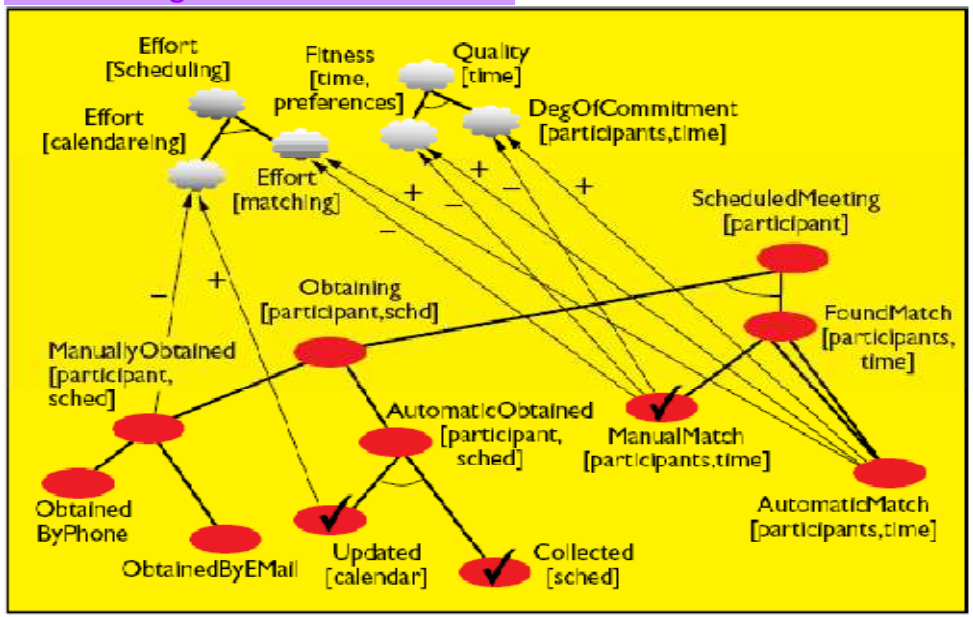
Softgoal Interdependency Graph (SIG)

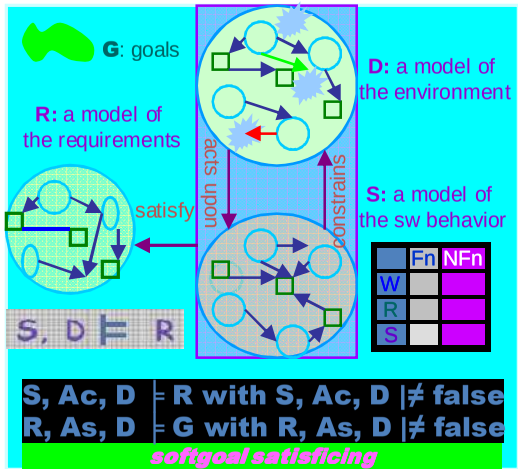


Object Model



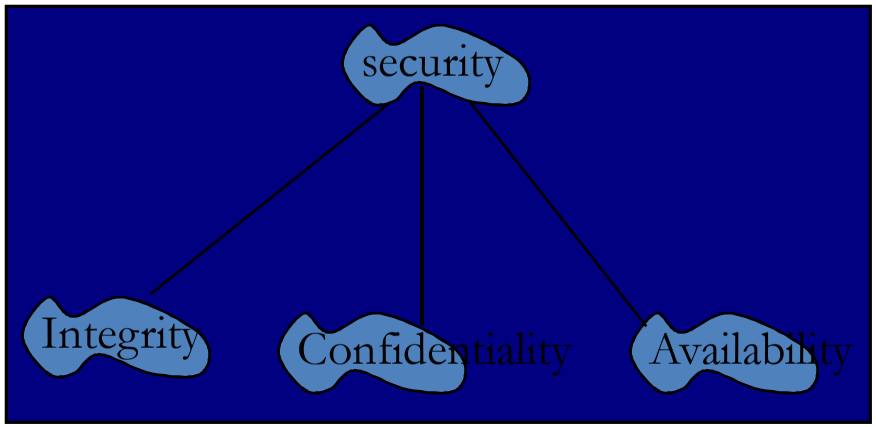
From Softgoals to Use Cases

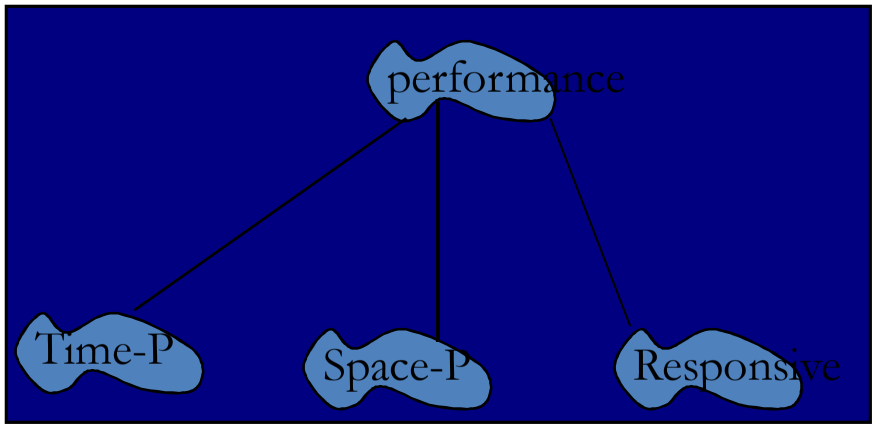




NFRs:
subjective in both *definitions* & solutions

- Know at least what you mean - ***decompose***

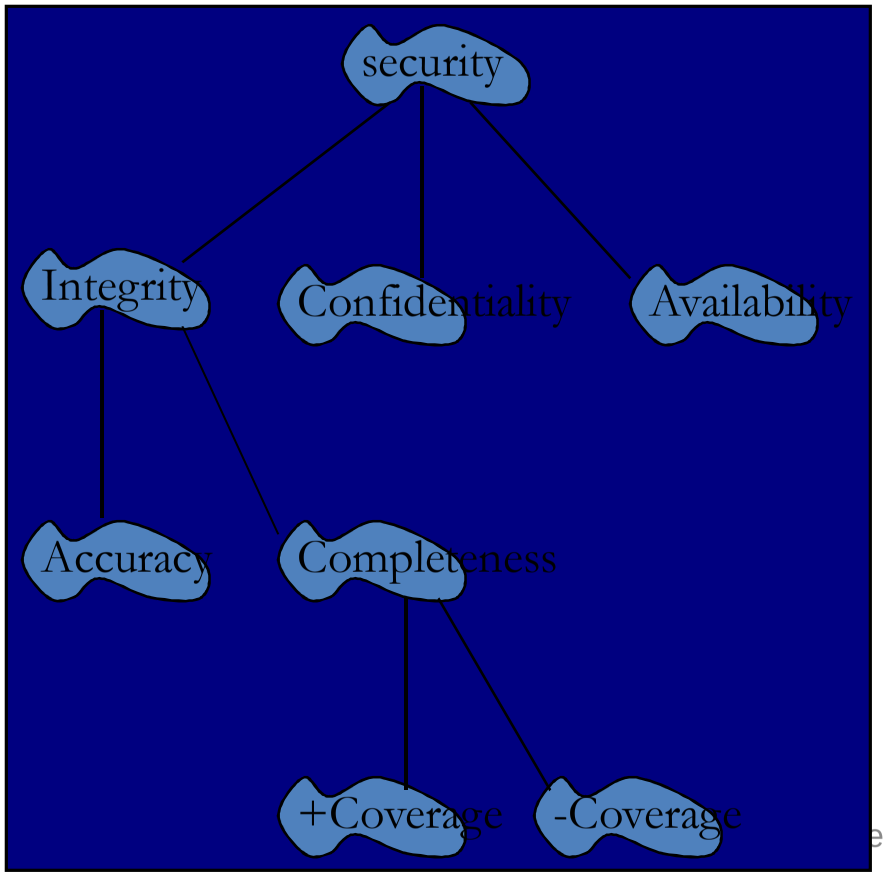


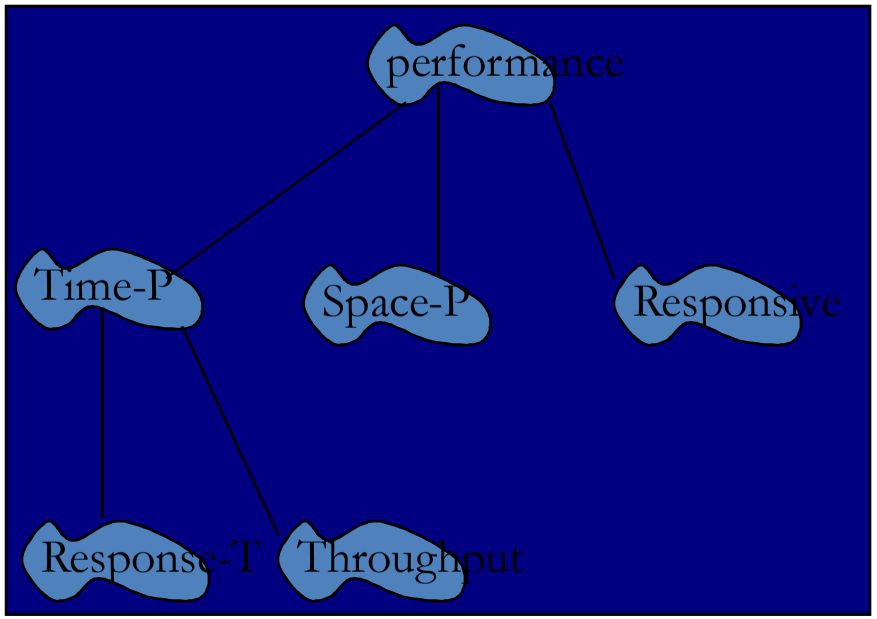


NFRs:
subjective in both *definitions* & solutions

- Know at least what you mean *as precisely as possible*

  - *as many decompositions as needed*

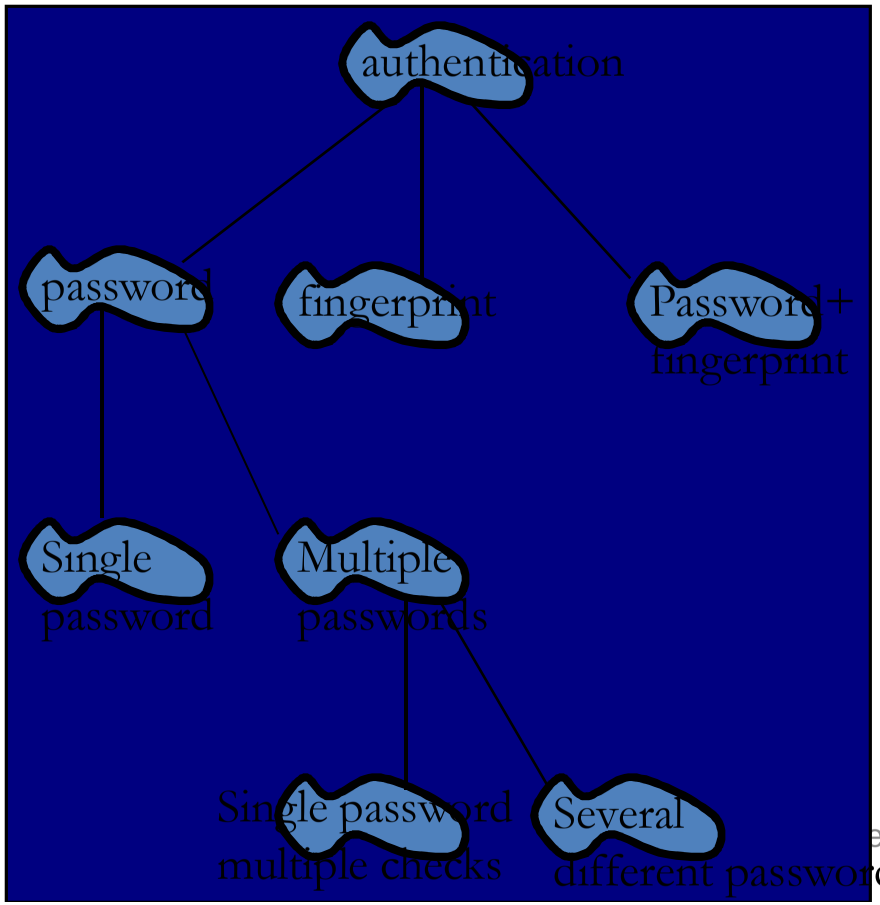


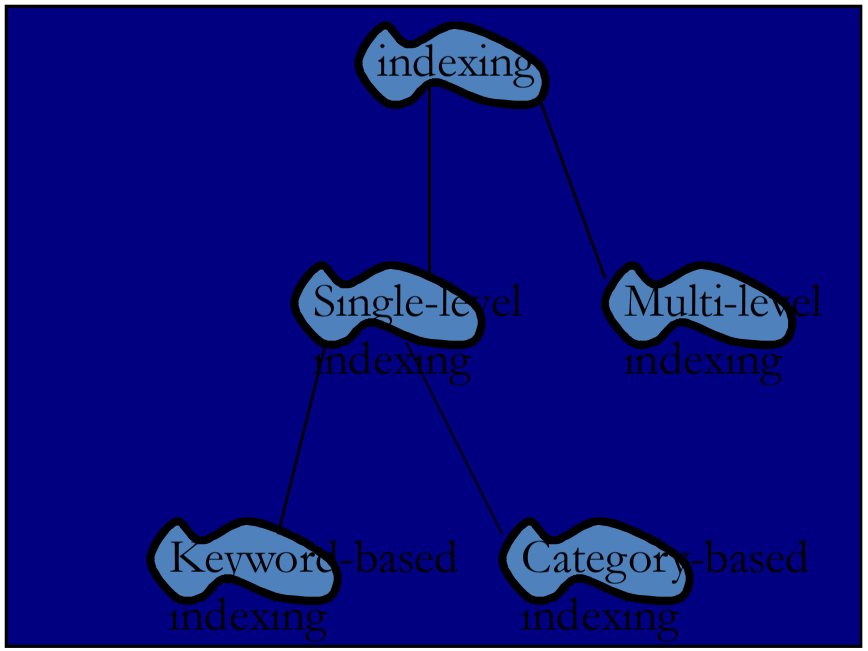


NFRs:
subjective in both definitions & *solutions*

- Know at least what you mean *as precisely as possible*

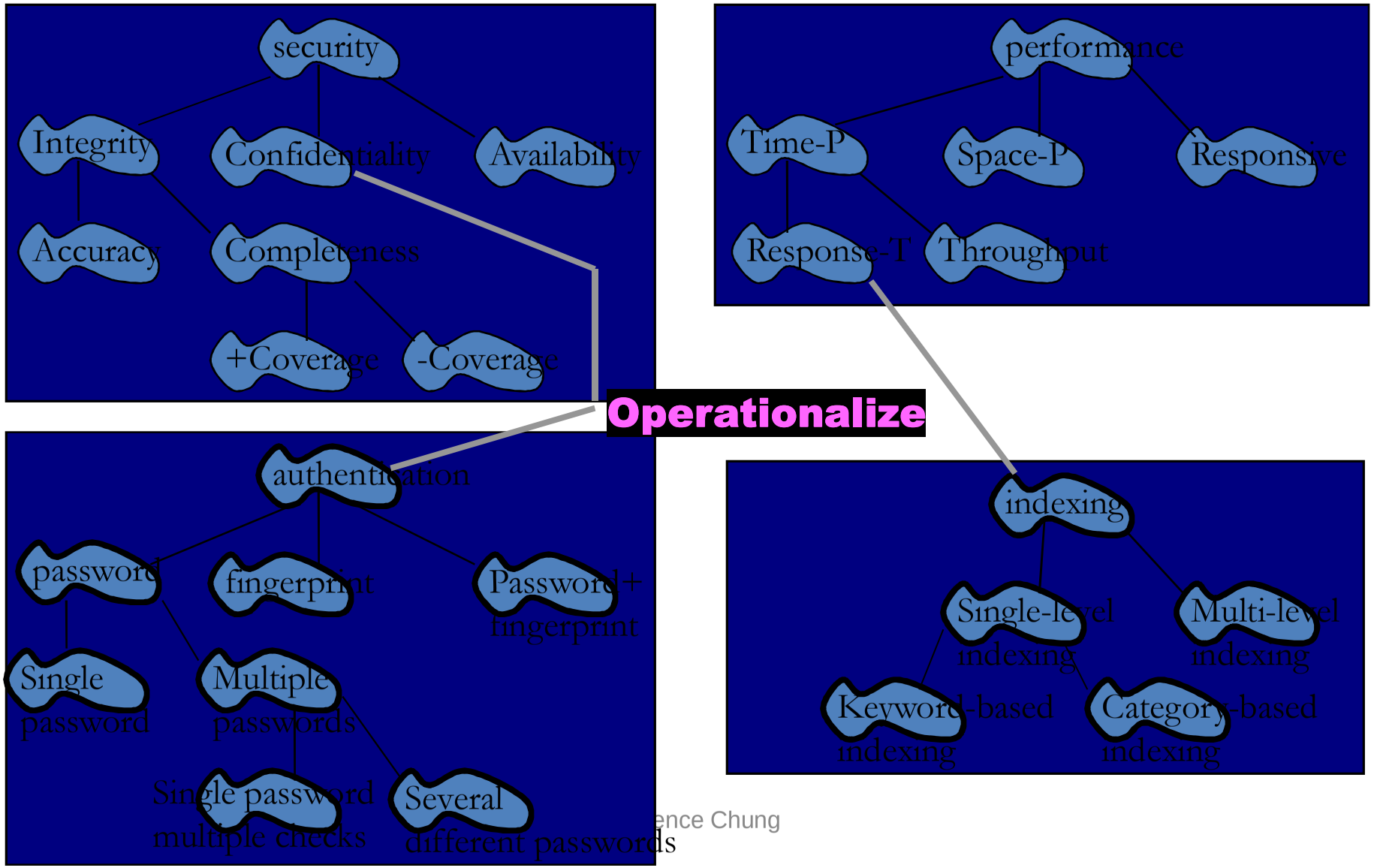
  - as many decompositions as needed





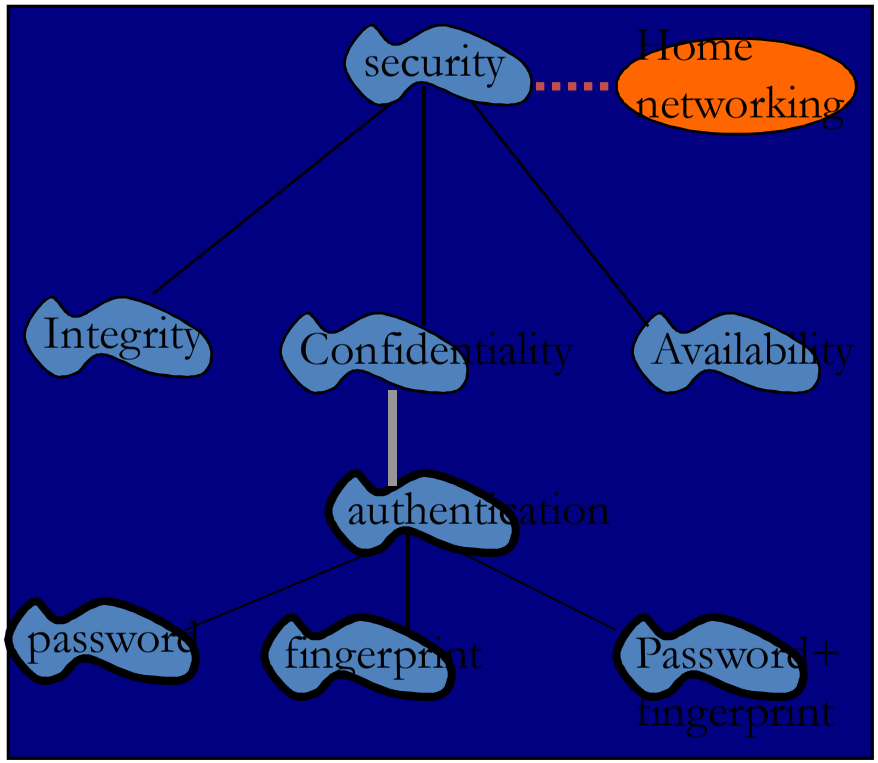
NFRs:
subjective in both *definitions* & *solutions*

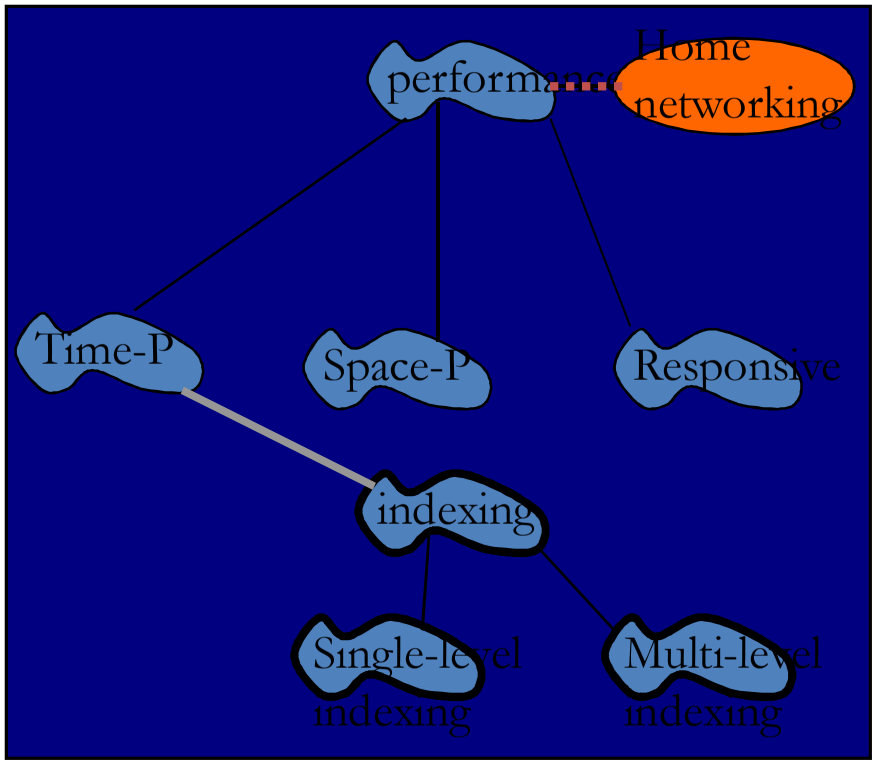
- Know at least what you mean *as precisely as possible* - *as many decompositions as needed*



NFRs:
non-functional …and…functional

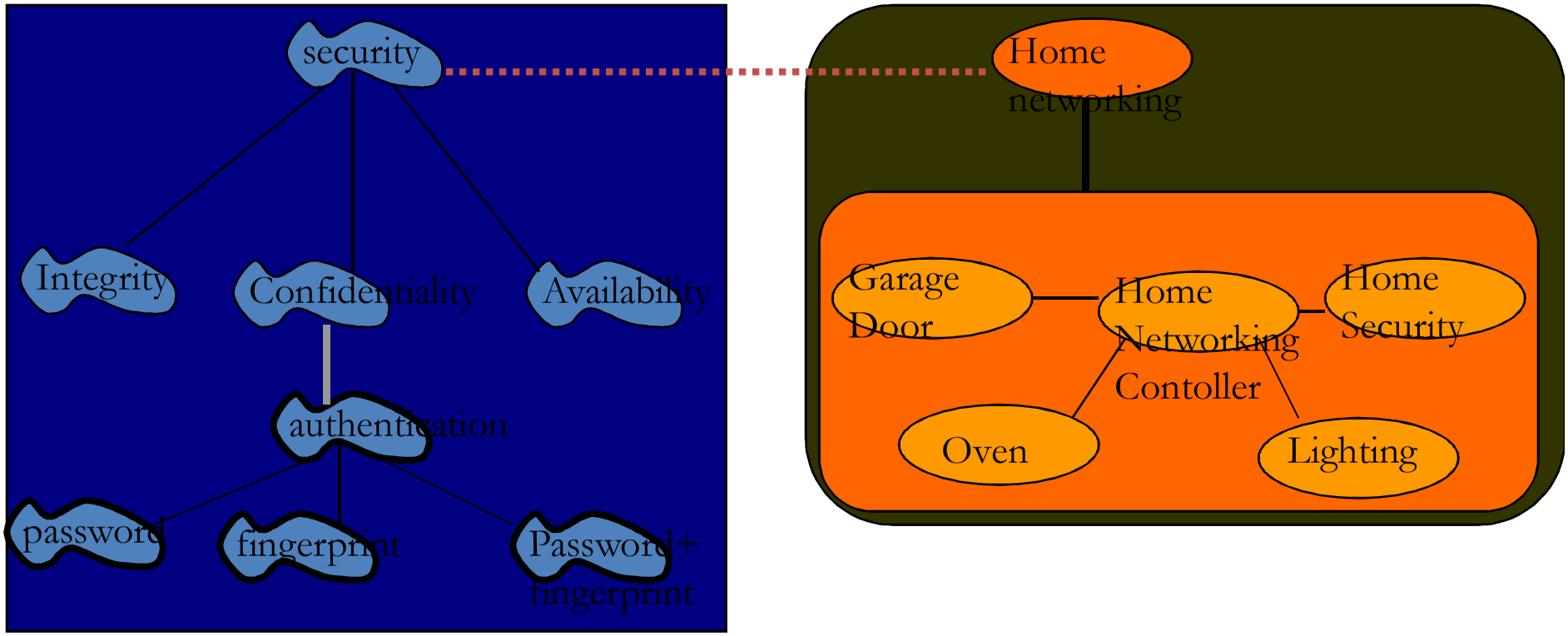
- Know at least what you mean – decompose
- **Relate Functional and Non-functional sides**





NFRs:
non-functional …and…functional

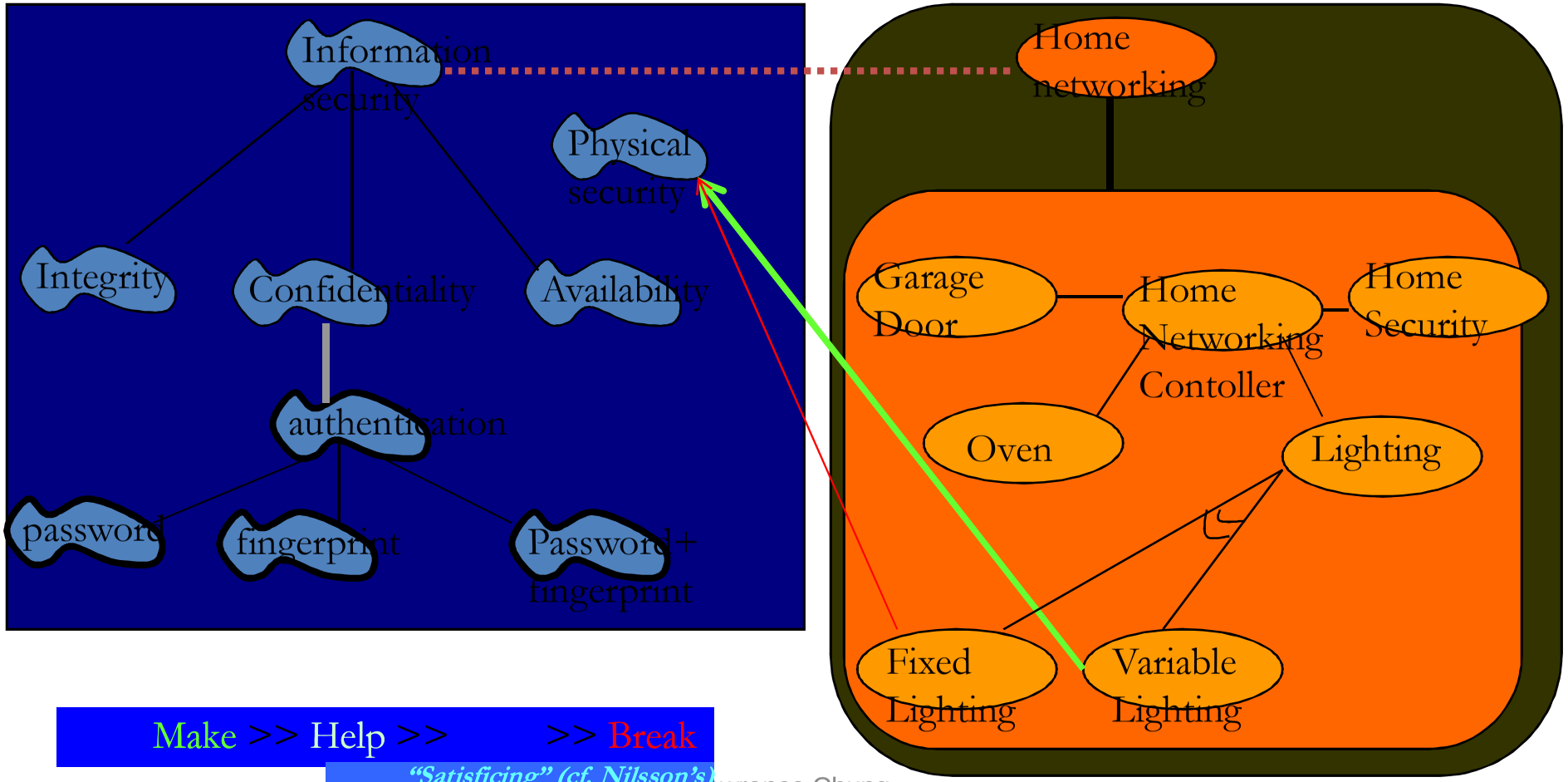
- Know at least what you mean – decompose
- Relate Functional and Non-functional sides
- ***Be as specific about the scope/topic/parameter: from global to local***



- Security ⇨ Security [Home Networking] ⇨ Security [Garage Door, Home Networking]
- Authentication ⇨ authentication [Home Networking] ⇨ hentication [Garage Door, Home Networking]

NFRs:
non-functional …and…
subjective in both definitions & solutions

- Know at least what you mean – decompose
- Relate Functional and Non-functional sides
- ***Different functional operationalizations contribute differently***

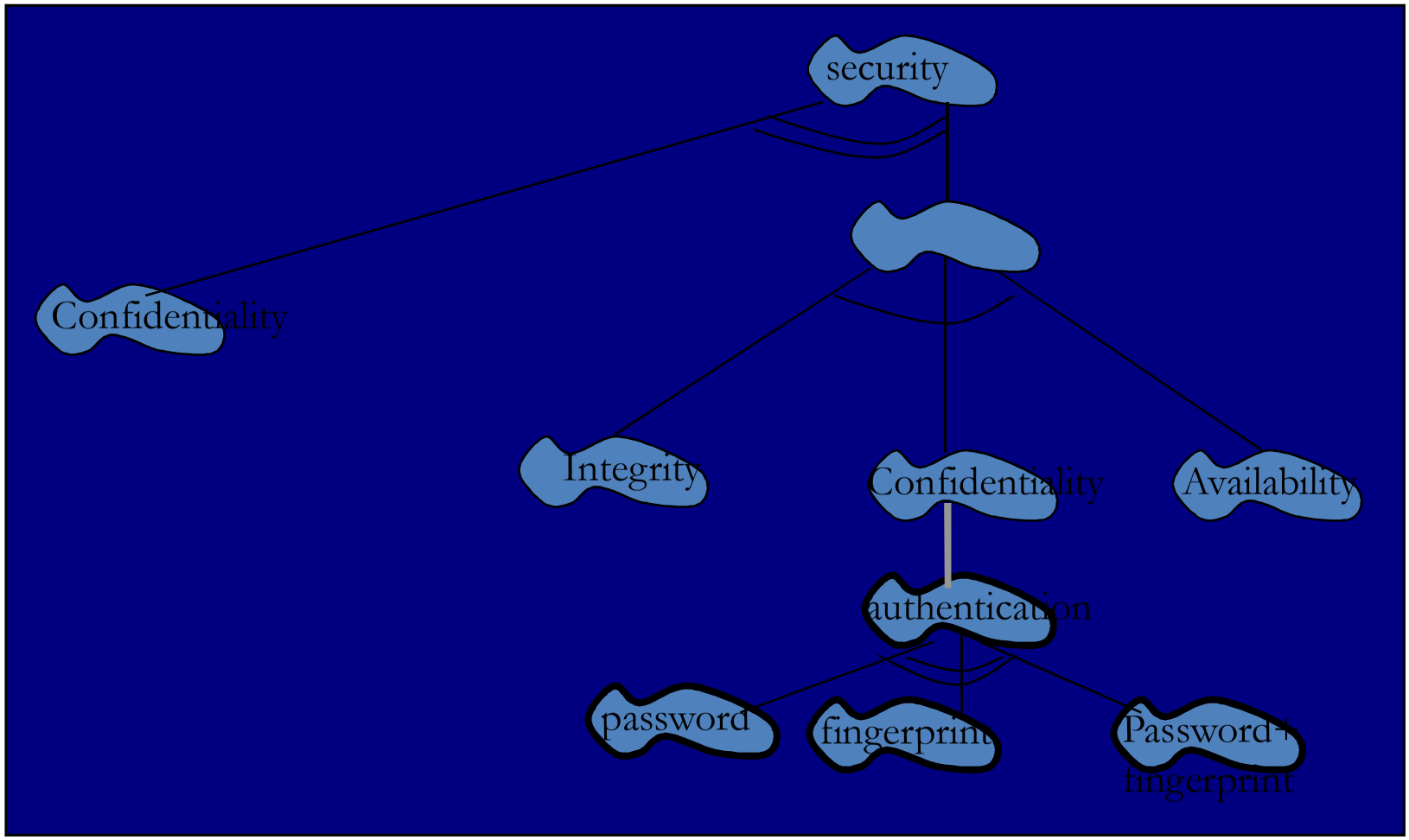


Make >> Help >> >> Break

"Satisficing" (cf. Nilsson's)

NFRs:
graded in both definitions and solutions – and relative

- **Explore alternatives** – some are better/worse than others

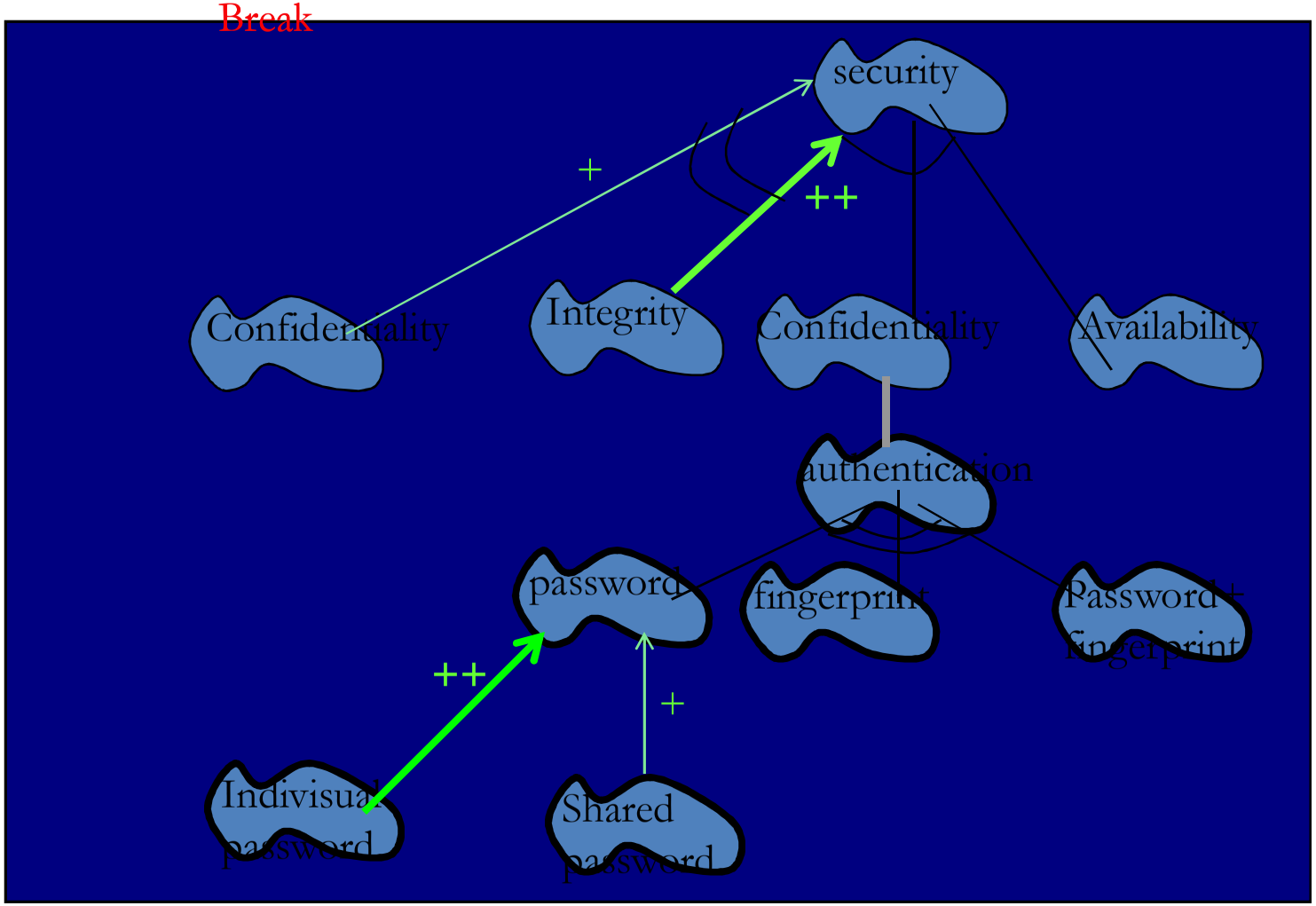


NFRs:
graded in both definitions and solutions – and relative

- Explore alternatives – some are better/worse than others
- ***Different alternatives may have different degrees of contributions***

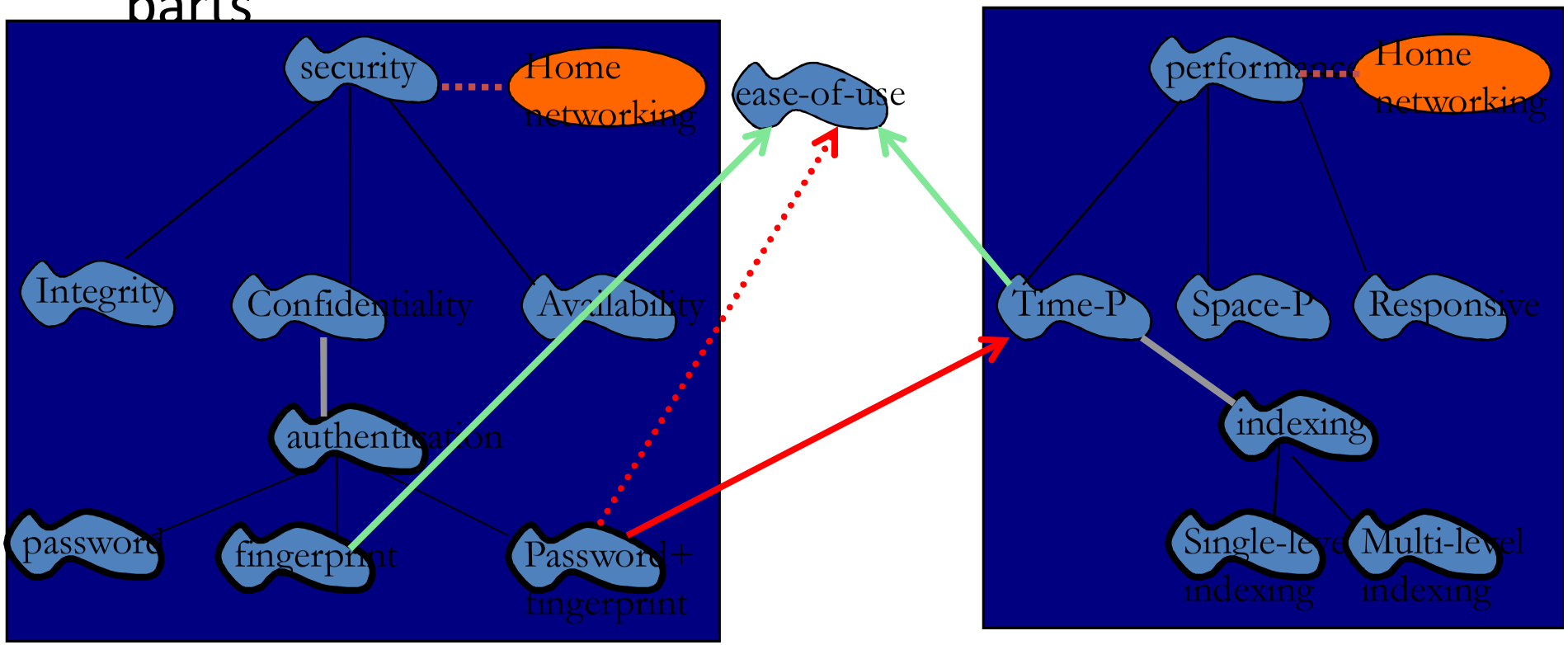
Make >> Help >> >> *"Satisficing" (cf. Nilsson's)*

~~Break~~



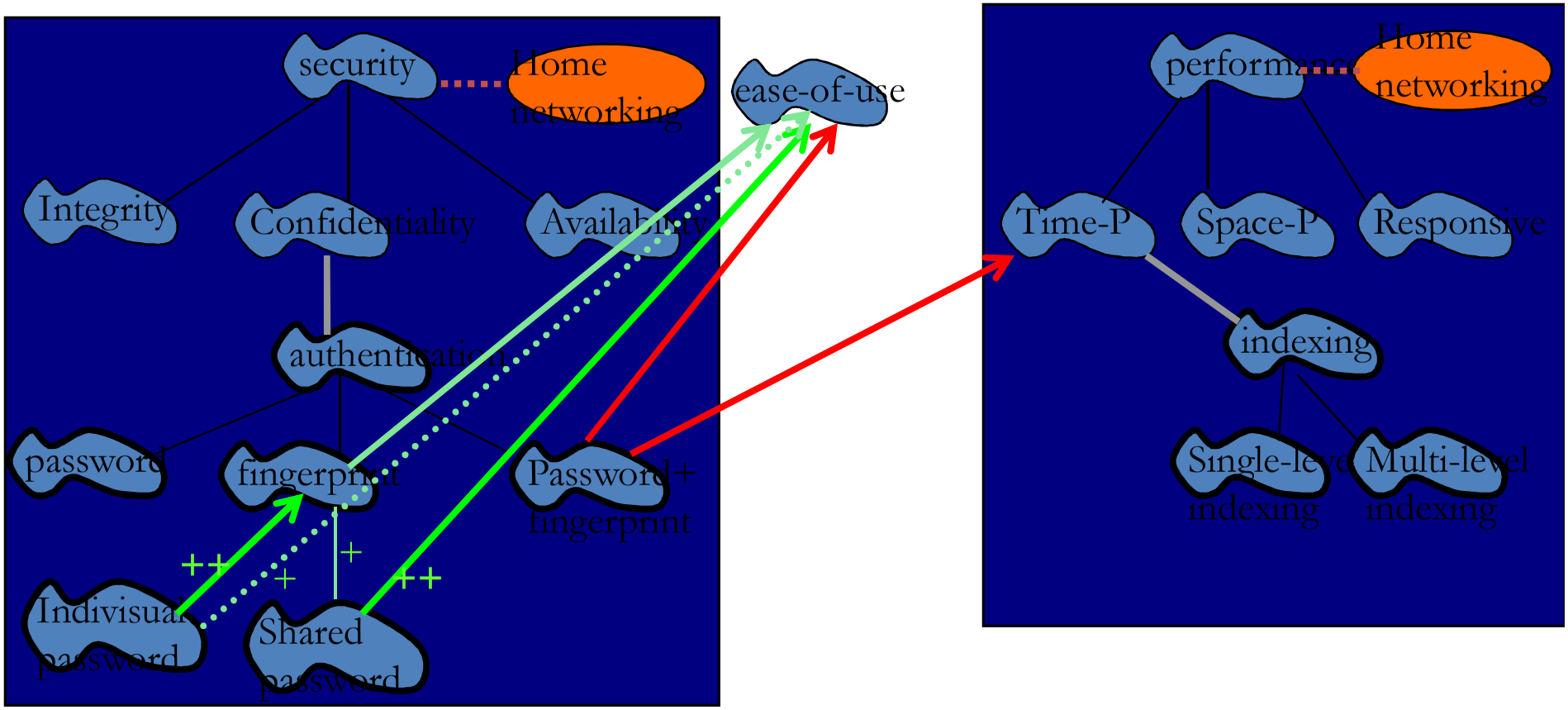
NFRs: interacting

- Conflicting: the whole is less than the sum of its parts
- Synergistic: the whole is greater than the sum of its parts



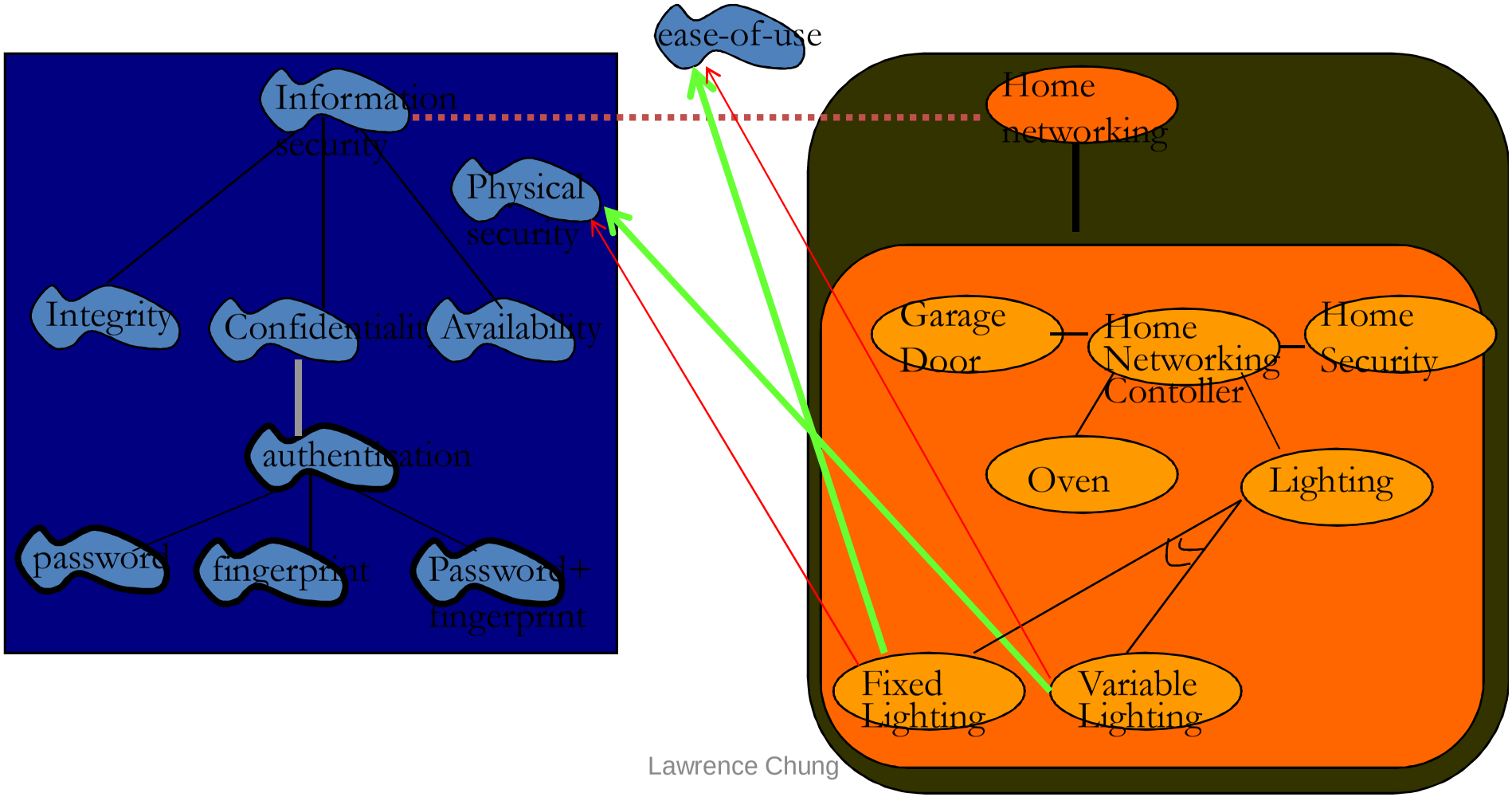
NFRs:
interacting – graded/relative

– Different techniques thru ***nfr-operationalizations*** have different impacts (cf. fr-operationalizations)



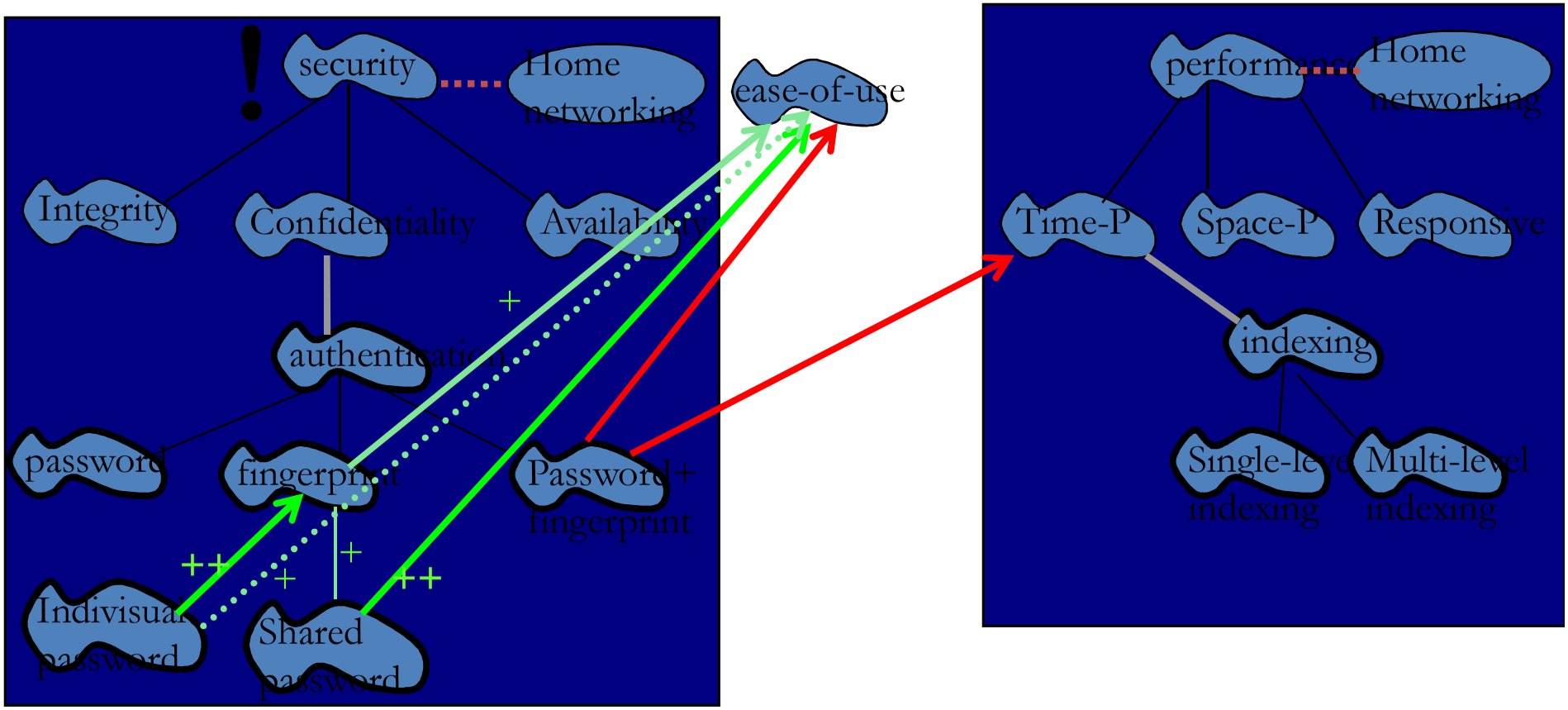
NFRs:
interacting – graded and relative

– Through functional choices (***fr-operationalizations***)



NFRs:
interacting – graded/relative

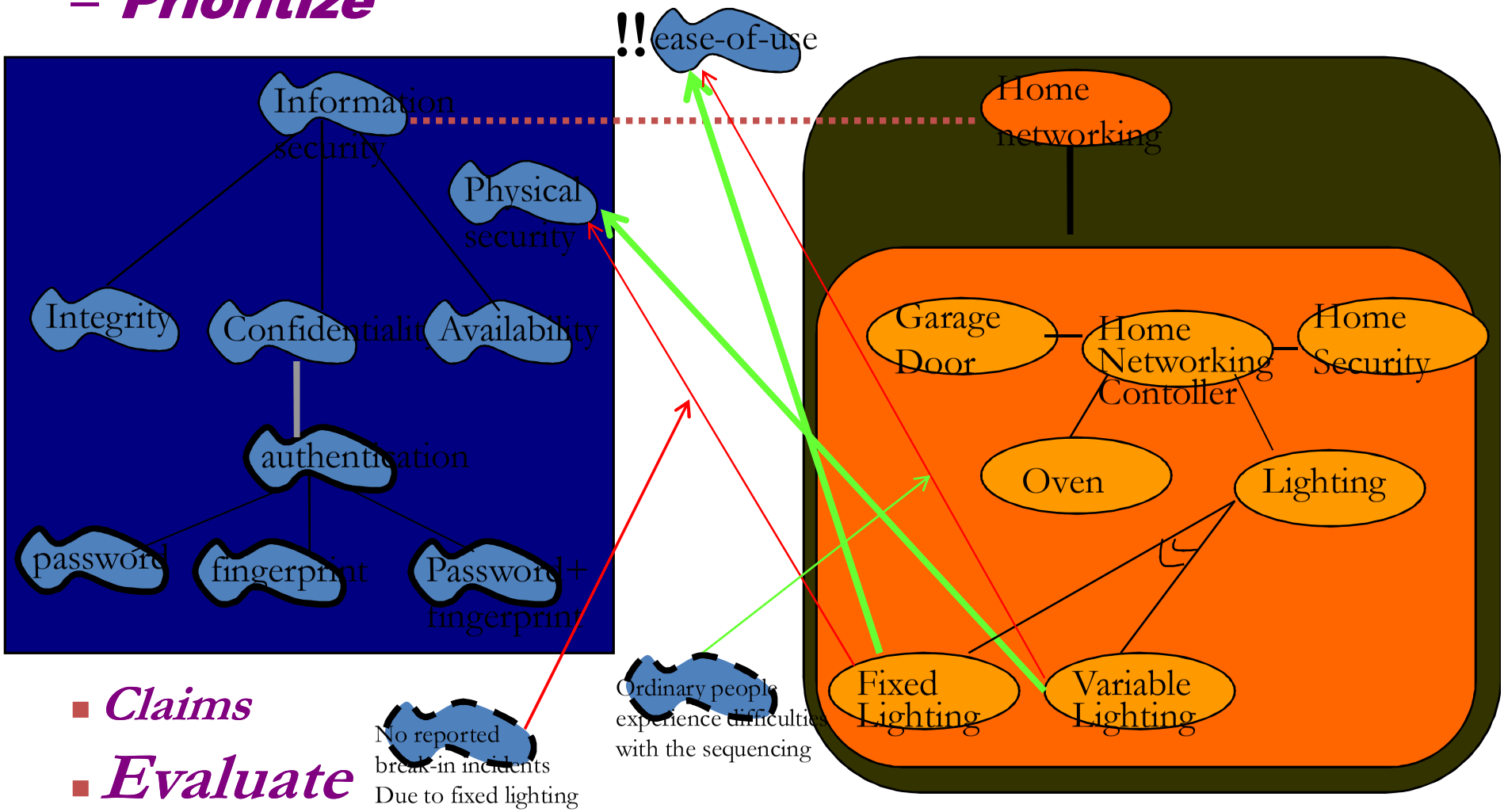
- Different techniques have different impacts
- ***Prioritize***



NFRs:
interacting – graded and relative

– Through functional choices

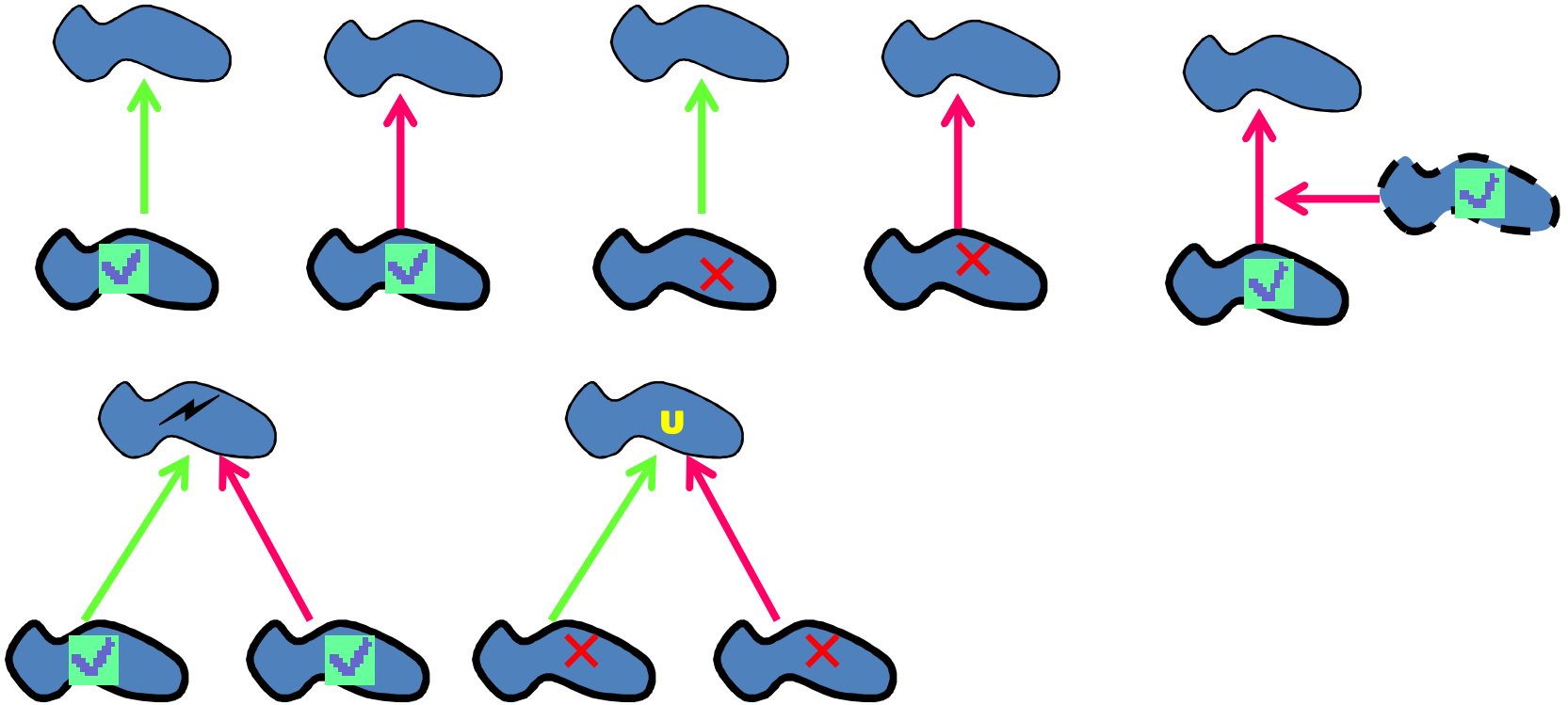
– ***Prioritize***

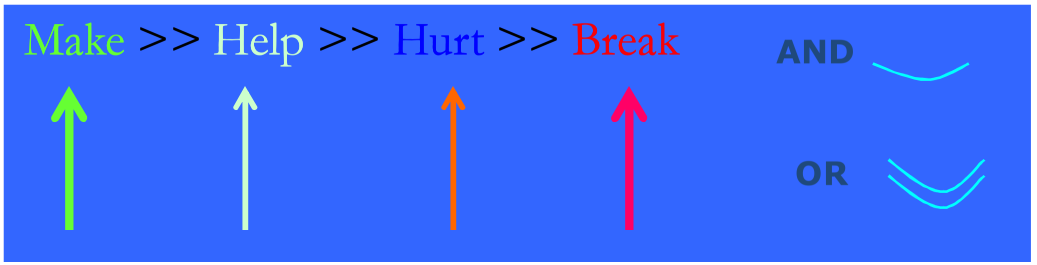


- ***Claims***
- ***Evaluate***

thru propagation of labels (satisficed, denied)

Softgoal Interdependency Graph (SIG): Evaluation Thru Label Propagation

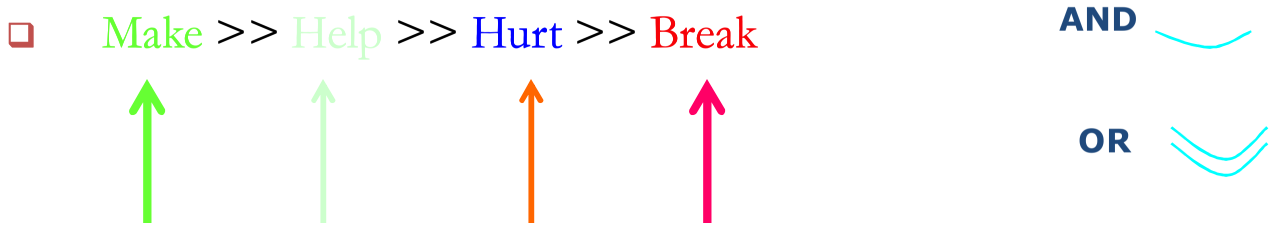




Softgoal Interdependency Graph (SIG): Summary of Modeling Concepts

- Softgoals: NFR Softgoals, Operationalizing Softgoals, Claim Softgoals

- Contributions:



- "Satisficing"

Softgoal Interdenpency Graph (SIG): Semantics

- Proposition = Softgoal U Contribution

$$AND: \quad Propositions \times 2^{Propositions}.$$

$$satisficed(G_1) \wedge satisficed(G_2) \wedge \ldots \wedge satisficed(G_n) \ \wedge$$
$$satisficed(AND(G_0, \{G_1, G_2, \ldots, G_n\})) \longrightarrow satisficeable(G_0)$$

$$(denied(G_1) \vee denied(G_2) \vee \ldots \vee denied(G_n)) \ \wedge$$
$$satisficed(AND(G_0, \{G_1, G_2, \ldots, G_n\}) \longrightarrow deniable(G_0)$$

$$OR: \quad Propositions \times 2^{Propositions}.$$

$$denied(G_1) \wedge denied(G_2) \wedge \ldots \wedge denied(G_n) \ \wedge$$
$$satisficed(OR(G_0, \{G_1, G_2, \ldots, G_n\})) \longrightarrow deniable(G_0)$$
$$(satisficed(G_1) \vee satisficed(G_2) \vee \ldots \vee satisficed(G_n)) \ \wedge$$
$$satisficed(OR(G_0, \{G_1, G_2, \ldots, G_n\}) \longrightarrow satisficeable(G_0)$$

Softgoal Interdenpency Graph (SIG): Semantics

MAKE $Propositions \times Propositions.$

$satisficed(G_1) \ \wedge \ satisficed(\text{MAKE}(G_0, G_1)) \longrightarrow satisficeable(G_0)$

BREAK $Propositions \times Propositions.$

$satisficed(G_1) \ \wedge \ satisficed(\text{BREAK}(G_0, G_1)) \longrightarrow deniable(G_0)$

HELP $Propositions \times Propositions.$

$denied(G_1) \ \wedge \ satisficed(\text{HELP}(G_0, G_1)) \longrightarrow deniable(G_0)$

If *satisficed(HELP(G_0, G_1))* then there exist propositions G_2, …, G_n such that

$\neg(satisficed(G_2) \wedge \ldots \wedge satisficed(G_n) \longrightarrow satisficeable(G_0))$

but

$satisficed(G_1) \wedge satisficed(G_2) \wedge \ldots \wedge satisficed(G_n) \ \wedge \ satisficed(\text{HELP}(G_0, G_1))$

$\longrightarrow satisficeable(G_0)$

HURT $Propositions \times Propositions.$

$denied(G_1) \ \wedge \ satisficed(\text{HURT}(G_0, G_1)) \longrightarrow satisficeable(G_0)$

$If \ \text{HURT}(G_0, G_1) \ then \ there \ exist \ G_2, \ldots, G_n \ such \ that$

$\neg(satisficed(G_2) \wedge \ldots \wedge satisficed(G_n) \ \wedge \ satisficed(\text{HURT}(G_0, G_1))$

$\longrightarrow deniable(G_0))$

but

$satisficed(G_1) \wedge satisficed(G_2) \wedge \ldots \wedge satisficed(G_n) \ \wedge$

$satisficed(\text{HURT}(G_0, G_1)) \longrightarrow deniable(G_0)$

$und: \quad Propositions \times Propositions.$

Und(G_0, G_1) = MAKE(G_0, G_1) ∨ HELP(G_0, G_1) ∨ HURT(G_0, G_1) ∨ BREAK(G_0,G_1)

$eql: \quad Propositions \times Propositions.$

Softgoal Interdenpency Graph (SIG): Process of Construction

An iterative, interleaving process!!!

- Post NFR Softgoals:
- Refine NFR Softgoals as many times until the meaning is clear
  - Refine the type
  - Refine the topic list
  - Refine the priority

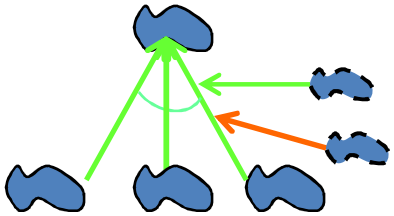
- Operationalize NFR Softgoals
- Refine Operationalizing Softgoals as many times until all the parts and relationships are designed (N.B: recall "one person's floor is another person's ceiling")
  - Refine the type
  - Refine the topic list
  - Refine the priority

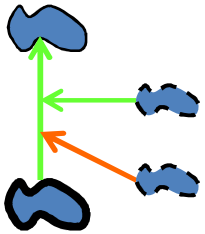
- Provide justifications in terms of Claim Softgoals, for any kind of softgoals
- Evaluate the degree to which each softgoal is satisficed.

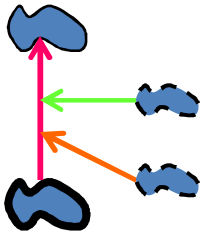
The NFR Framework:

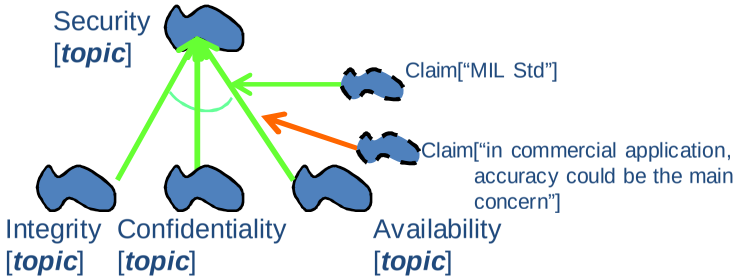
Reuse of Knowledge of NFRs

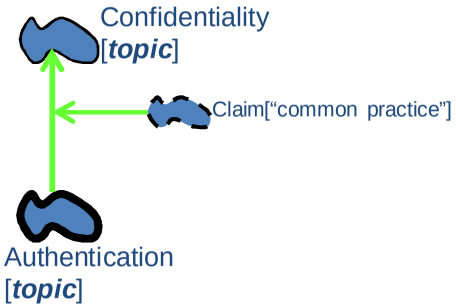
- Introduces ***Catalogues*** of NFRs much like patterns for design are built
  - Methods:
  - Correlation Rules:

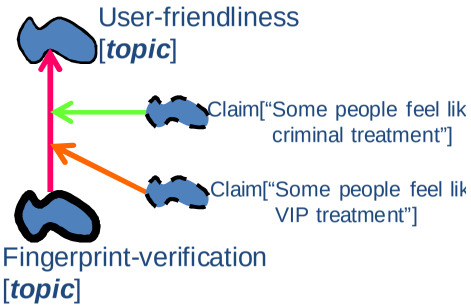


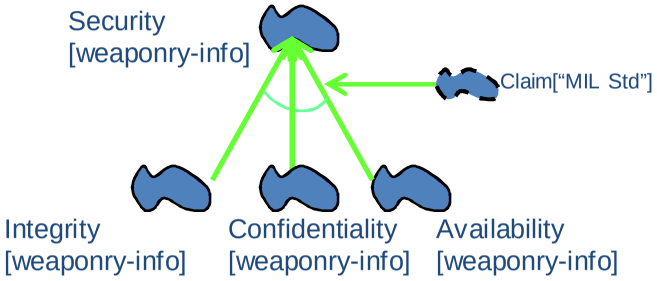


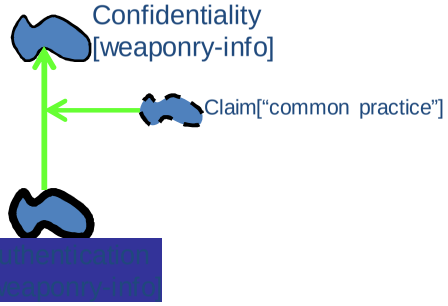


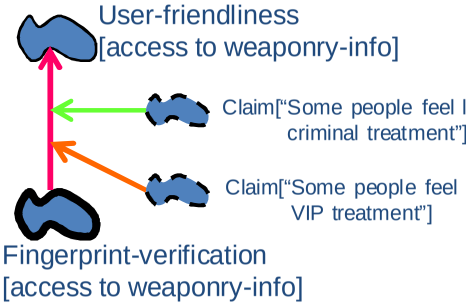












The NFR Framework

{Chung et.al.} Non-functional Requirements in Software Engineering [Software Architecture lecture notes]

- Patterns:

The NFR Framework

and the Reference Model

Recall: Example 1: Patient Monitoring

D_1: There will always be a nurse close enough to hear the buzzer

D_2: The sound from the heart falling below a certain threshold indicates that heart has (is about to) stop

R_1: A warning system notifies the nurse if the patient's heartbeat stops

S_1: If the sound from the sensor falls below a certain threshold, the buzzer shall be actuated.

C – with a microphone as a sensor and a buzzer as an actuator

P - Program

Designation Categories:

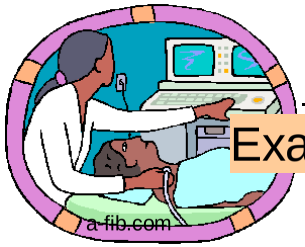
e_h: the nurse and the heartbeat of the patient.

e_v: sounds from the patient's chest.

s_v: the buzzer at the nurse's station.

s_h: internal representation of data from the sensor.

Recall:



The NFR Framework

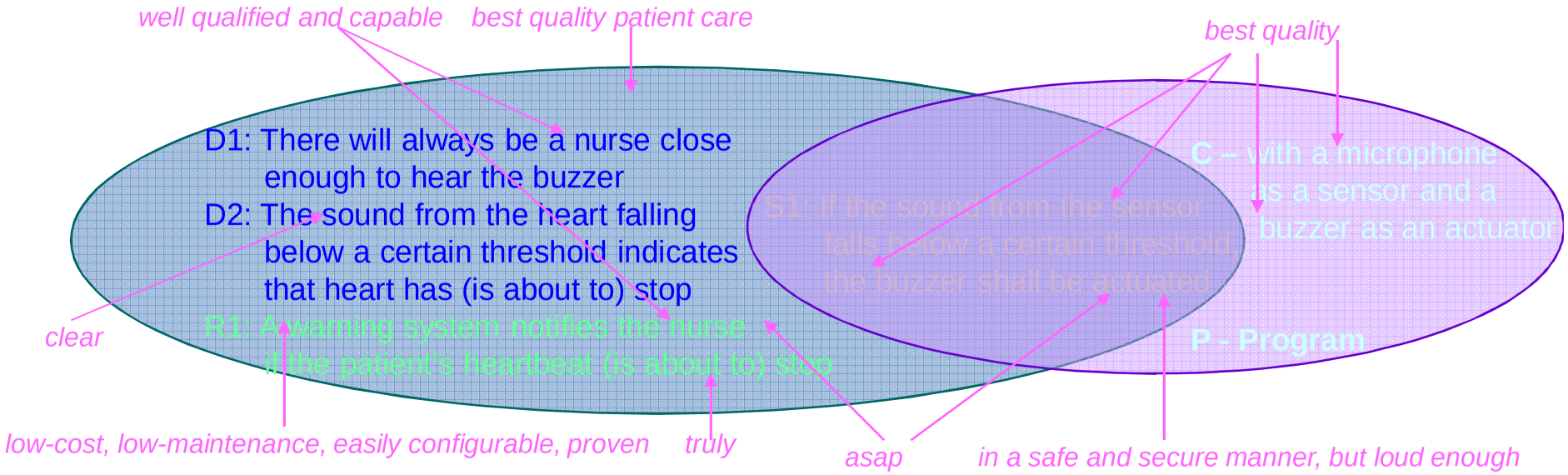
and the Reference Model

Example 1: Patient Monitoring

Need: monitoring if a patient's heart is failing

Problem: monitoring if a patient's heart is failing is difficult and sometimes has been unsuccessful

- ***A nurse cannot stay close to the patient always and on alert***



well qualified and capable best quality

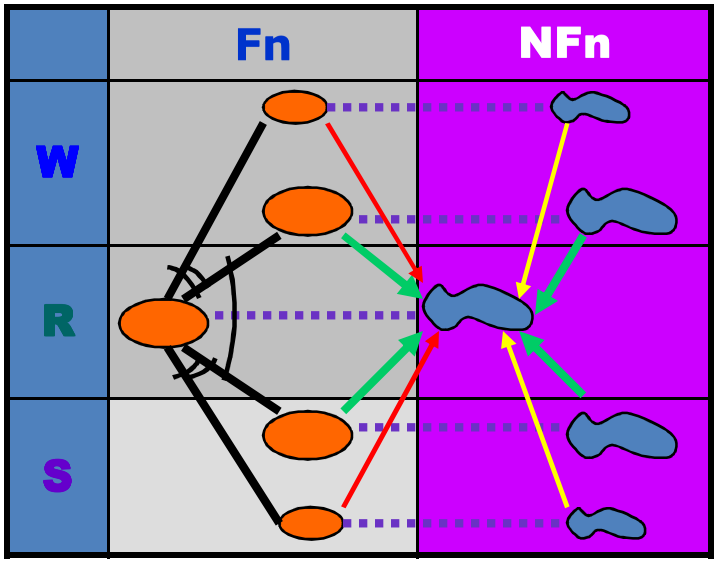
Designation Categories:

e_h: the nurse and the heartbeat of the patient.
e_v: sounds from the patient's chest.
s_v: the buzzer at the nurse's station.
s_h: internal representation of data from the sensor.

The NFR Framework

in relation to the Reference Model, KAOS, Tropos

- **WRSPM:** $S, D \models R$
- **KAOS:** $S, Ac, D \models R$ with $S, Ac, D \not\models$ false
  $R, As, D \models G$ with $R, As, D \not\models$ false
- **The NFR Framework:**



- *nfr-operationalizations*
- fr-operationalizations

- *Any phenomena/functional description, indicative or optative or expectational, and any agent can be associated with softgoals*
- satisficed (Q(S^G)), satisficed (Q(D^G)) ⊨ satisficed (Q(R^G))
- satisficed (Q(P^G)), satisficed (Q(M^G)) ⊨ satisficed (Q(S^G))

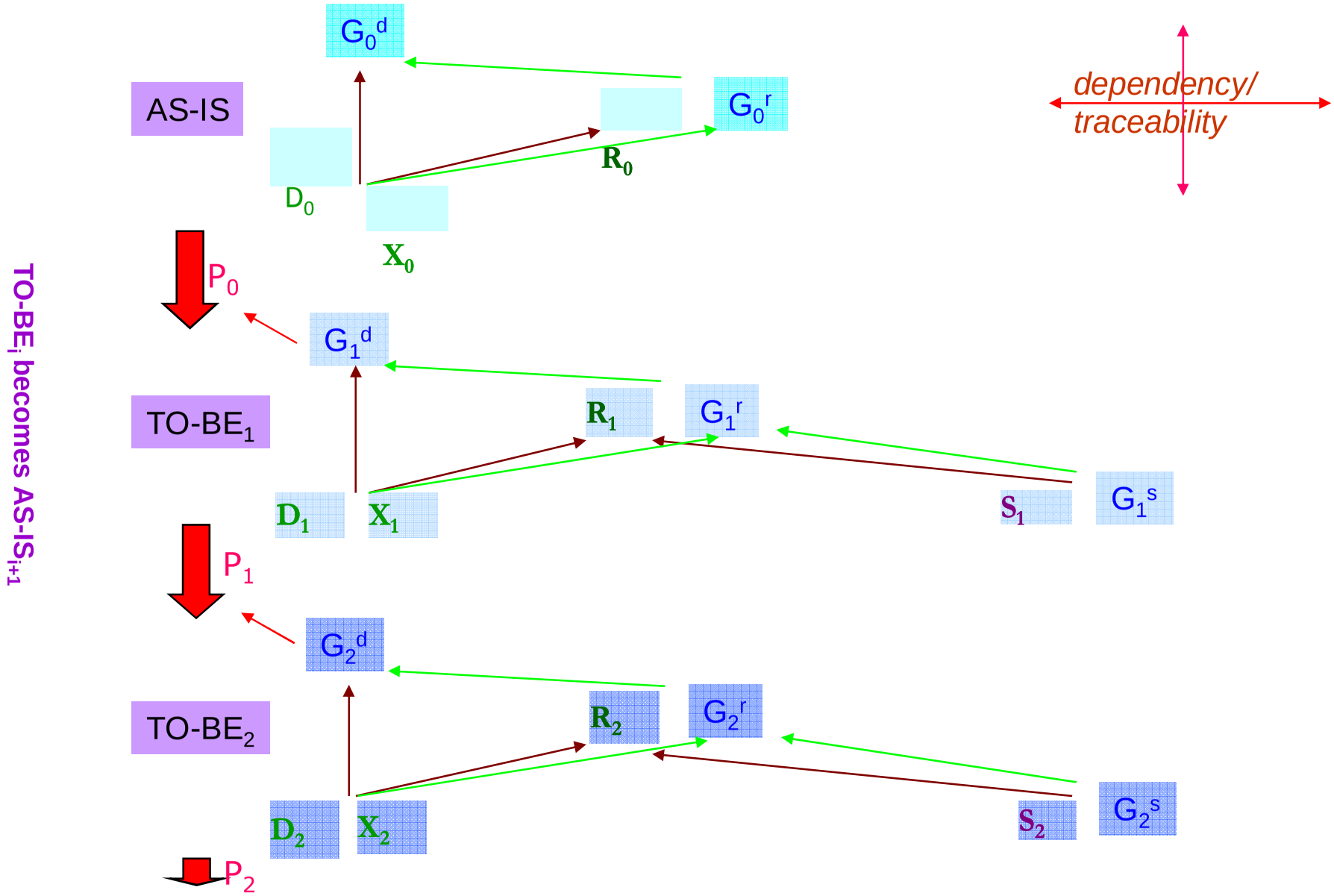
$M^G, Prog^G \models S^G; S^G, D^G \models R^G; R^G, D^G \models G; (G \models \neg P) \vee (G \sim \neg P)$

What the Metaphysics of Quality would do is take this separate category, Quality, and show how it contains within itself both subjects and objects. The Metaphysics of Quality would show how things become enormously more coherent--fabulously more coherent--when you start with an assumption that Quality is the primary empirical reality of the world.... [Robert Pirsig]

Recall

Property Preserving Evolution

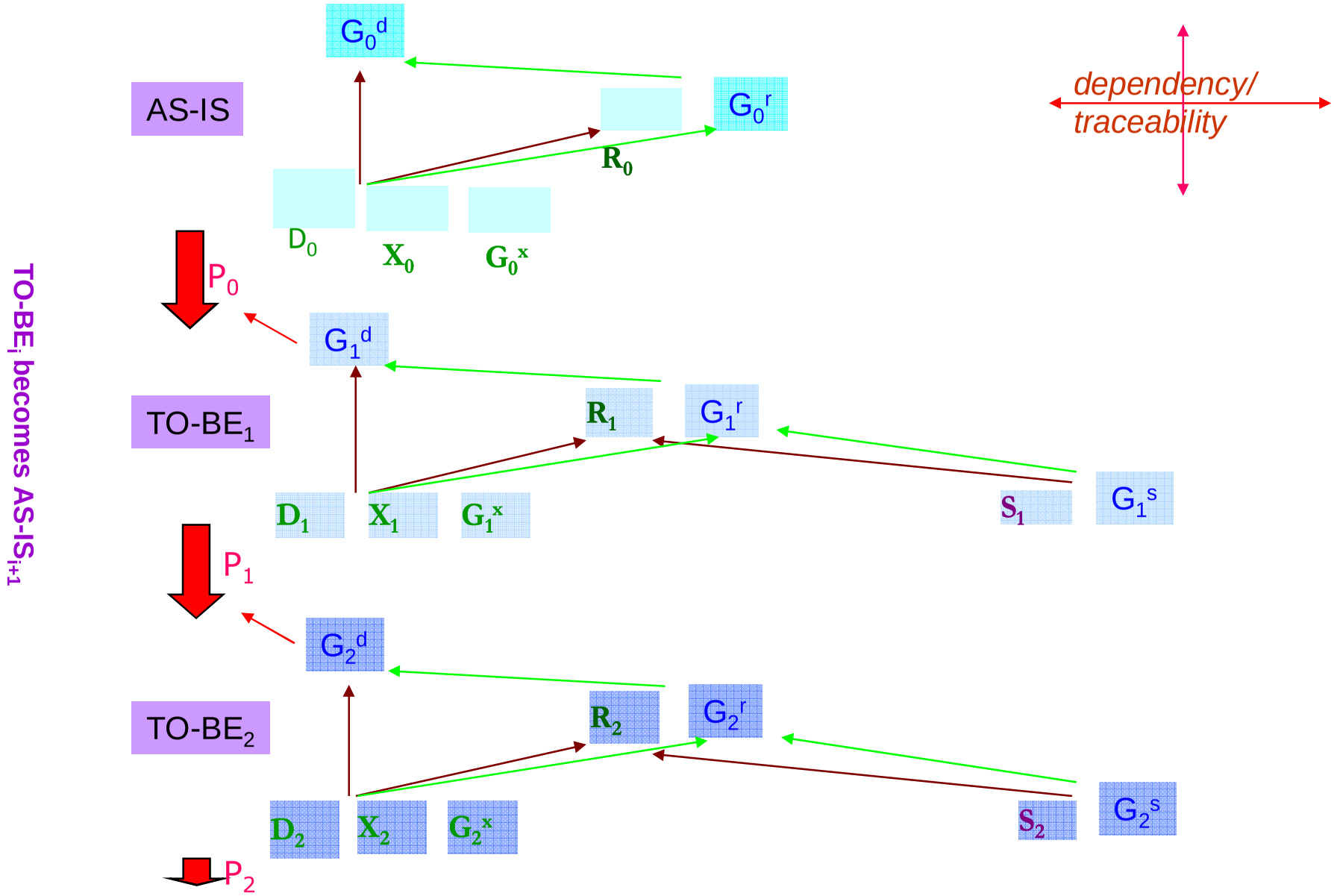
$(G_i^s, S_i), (D_i, X_i) \models R_i$; $(G_i^s, S_i), (D_i, X_i) \models G_i^r$; $(G_i^r, R_i), (D_i, X_i) \models G_i^d$



Recall

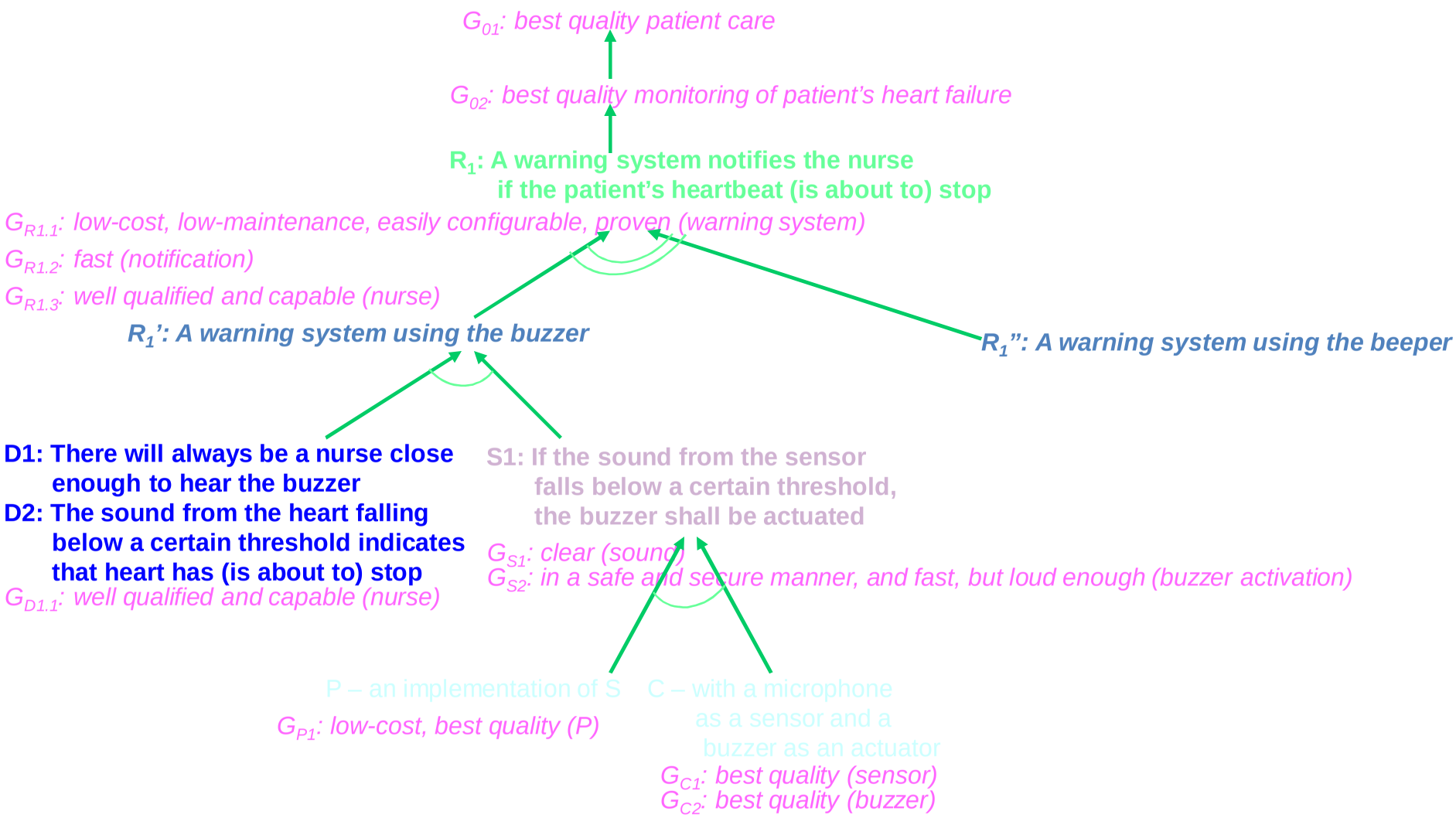
Property Preserving Evolution

$(G_i^s, S_i), (D_i, X_i, G_i^x) \models R_i$; $(G_i^s, S_i), (D_i, X_i, G_i^x) \models G_i^r$; $(G_i^r, R_i), (D_i, X_i, G_i^x) \models G_i^d$



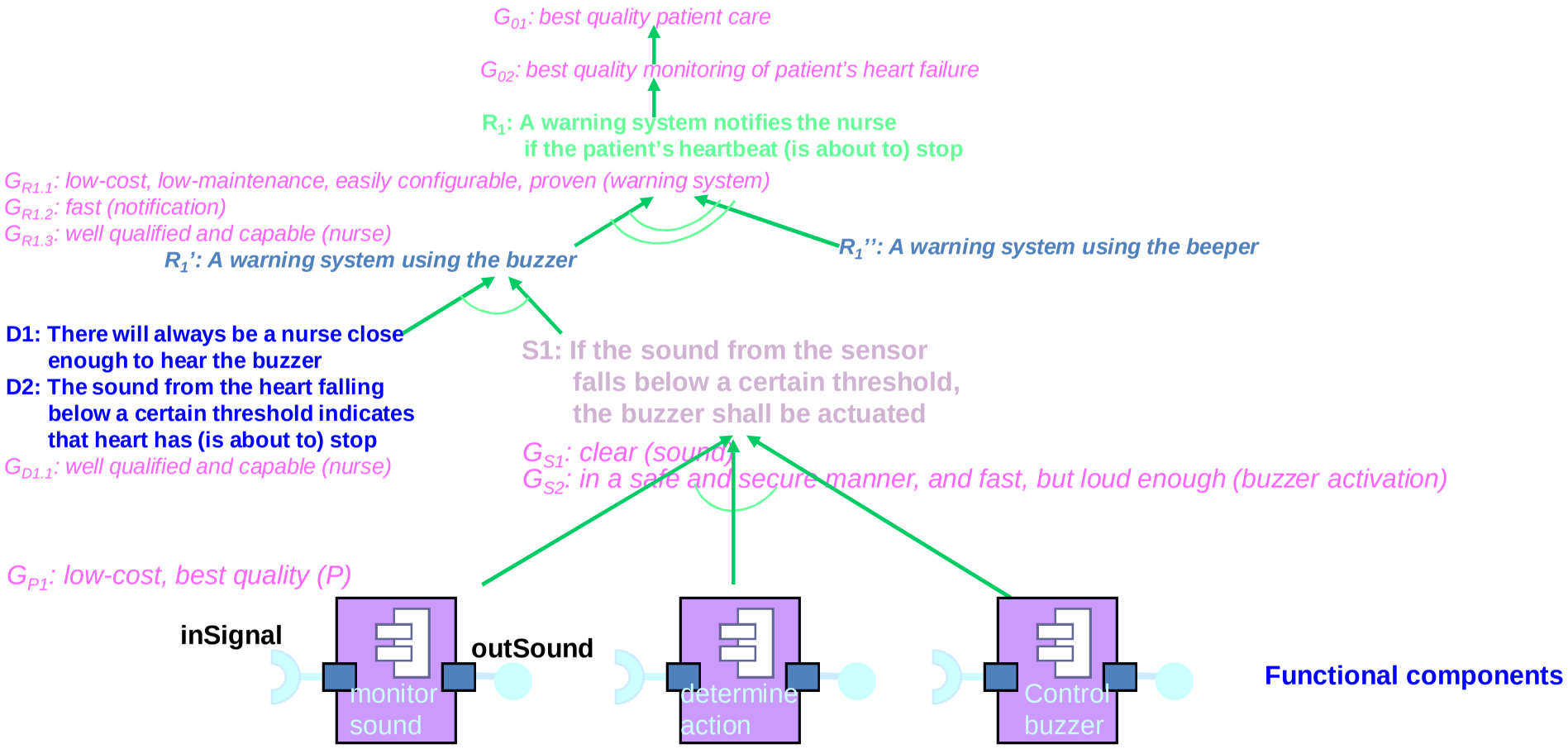
The NFR Framework

and the Reference Model



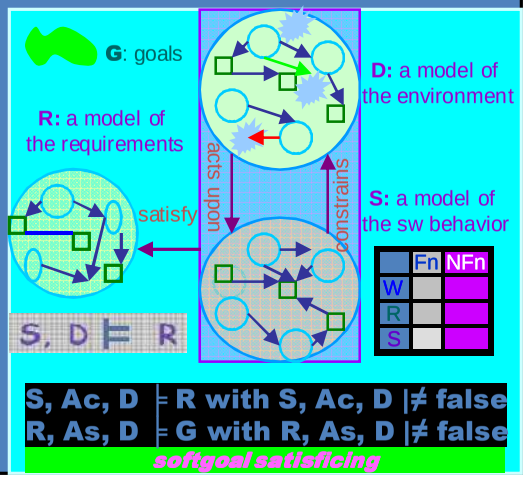
The NFR Framework

From Specification to Architecture



Non-Functional Requirements

How 2 – Dos and Don’ts



NFRs – Dos & Don'ts

➢ ***Dos***

- Relate to FRs
- Clarify scope/topic
- Identify agents, whenever useful
- Discover relationships between definitions of NFRs
- Discover relationships between solutions to NFRs
- Refine definitions as many times as needed
- Refine solutions as many times as needed
- Prioritize
- Discover conflicts
- Safeguard against conflicts
- Discover synergies
- Discover operationalizations as reasons for conflicts/synergies
- Determine strengths of contributions
- Justify strengths of contributions
- Explore alternatives
- Discover solutions from requirements
- Discover requirements from solutions
- Consider use of multiple solutions
- Consider scenarios
- If necessary, quantify
- Evaluate, …subjectively, …objectively
- Establish traceability

➢ ***Don'ts***

- Absolute security, absolute reliabilty, absolute safety, ….
- One definition fits all
- One solution solves all problems
- The contribution is such and such, since I say so
- Refine the definition only once
- They are falling down from the sky
- Dissociate from FRs
- May be more important than FRs, but should consume less resources
- You name it; our system does it
- No quantification, no existence
- Everybody needs the same
- Be only pessimistic
- Asking why "+" reveals ignorance
- Beg the question
- Evaluate & only evaluate
- Brainwash nothing but objectivity

Conflict resolution 1

- Delete email w. any zip file attachment

  -> misunderstanding betw. sender and receiver

  <- move email w. any zip file attachment into a junk file folder

  -> If the receiver does not check the junk file folder, still misunderstanding

  <- at the time the file is moved, notify this to the receiver

  -> if the receiver still does not check the junk file folder or checks it late, still misunderstanding

  <- at the time the file is moved, notify the sender too

  -> If the receiver checks the junk file folder and opens it and the file is an attack,

  still a security breach

  Delete email w. any zip file attachment and block any future email from the same sender

Conflict resolution 2

- If the receiver opens email w. zip file and the file is an attack, a security breach
- Delete email any w. zip file attachment

-> misunderstanding betw. sender and receiver

<- move email w. any zip file attachment into a junk file folder

-> If the receiver does not check the junk file folder, still misunderstanding

<- at the time the file is moved, notify this to the receiver

-> if the receiver still does not check the junk file folder or checks it late, still misunderstanding

<- at the time the file is moved, notify the sender too

-> If the receiver checks the junk file folder and opens it and the file is an attack, still a security breach

- Delete email w. any zip file attachment and block any future email from the same sender
- If the email is from a sender who is not in the list of allowed senders, delete it
- Leave the email but delete the attachment only

Conflict resolution 3

- Security[PC] -> S[email] -> S[sender] ^ S[recipient] ^ S[body] ^ S[attachment]
- Denied (S[attachment]) -> denied (S[email]) -> denied (S[PC])
- Zip(attachment] ^ attack(attachment) ^ open(attachment) -> denied (S[attachment])

  /* If the receiver opens email w. zip file and the file is an attack, a security breach */

  ~ Zip(attachment] v ~attack(attachment) v ~open(attachment) -> ~ denied (S[attachment]) helps ~denied(S[email])
- Delete email w. any zip file attachment

  -> misunderstanding betw. sender and receiver

  <- move email w. any zip file attachment into a junk file folder

  -> If the receiver does not check the junk file folder, still misunderstanding

  <- at the time the file is moved, notify this to the receiver

  -> if the receiver still does not check the junk file folder or checks it late, still misunderstanding

  <- at the time the file is moved, notify the sender too

  -> If the receiver checks the junk file folder and opens it and the file is an attack, still a security breach

- ❑ Leave the email, but delete the attachment only
- ❑ Leave the email, but delete the attachment only if it is an attack
- ❑ Leave the email but change the name of the attachment to "…renameToZip"
- ❑ If the email is from a sender who is not in the list of allowed senders, delete it
- ❑ Delete email w. any zip file attachment and block any future email from the same sender

Conflict resolution 4

- Security[PC] -> S[email] -> S[sender] ^ S[recipient] ^ S[body] ^ S[attachment]
- Denied (S[attachment]) -> denied (S[email]) -> denied (S[PC])
- Zip(attachment] ^ attack(attachment) ^ open(attachment) -> denied (S[attachment])

  /* If the receiver opens email w. zip file and the file is an attack, a security breach */

  ~ Zip(attachment] v ~attack(attachment) v ~open(attachment) -> ~ denied (S[attachment]) helps ~denied(S[email])
- Delete email w. any zip file attachment, at the time of reception

  -> misunderstanding betw. sender and receiver

  <- move email w. any zip file attachment into a junk file folder

  -> If the receiver does not check the junk file folder, still misunderstanding

  <- at the time the file is moved, notify this to the receiver

  -> if the receiver still does not check the junk file folder or checks it late, still misunderstanding

  <- at the time the file is moved, notify the sender too

  -> If the receiver checks the junk file folder and opens it and the file is an attack, still a security breach

- ❑ Leave the email, but delete the attachment only
- ❑ Leave the email, but delete the attachment only if it is an attack: detectable[attack(attachment)]
- ❑ Leave the email but change the name of the attachment to "…renameToZip"
- ❑ If the email is from a sender who is not in the list of allowed senders, delete it

NFRs – Where

➢ ***Wherever better/cheaper/faster/happier matters***

- Requirements Engineering
- System Architecting
- Software Architecting
- Design
- Implementation
- Validation & Verification
- Testing
- Maintenance
- Software Process
- Project Planning and Management
- Configuration Management
- Decision making

NFRs – How to represent

- *From informal to tabular to visual (a la html->xml->oo-xml/eb-xml/…; CRC cards->classes; use cases & use case templates)*

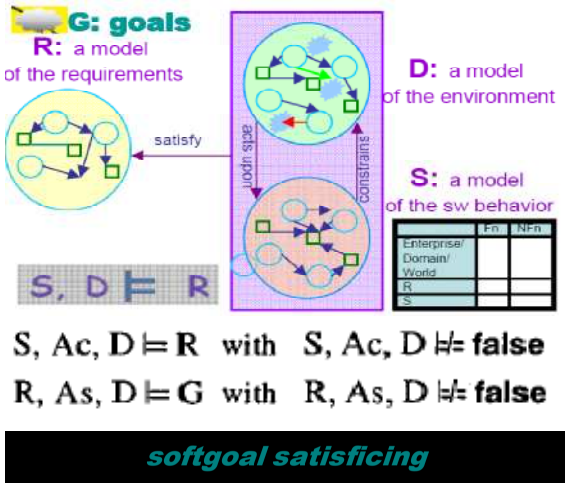
- ***Dos***

- Bring in FRs
- Clarify scope/topic
- Identify agents, whenever useful
- Discover relationships between definitions of NFRs
- Discover relationships between solutions to NFRs
- Refine definitions as many times as needed
- Refine solutions as many times as needed
- Prioritize
- Discover conflicts
- Safeguard against conflicts
- Discover synergies
- Discover operationalizations as reasons for conflicts/synergies
- Determine strengths of contributions
- Justify strengths of contributions
- Explore alternatives
- Discover solutions from requirements
- Discover requirements from solutions
- Consider use of multiple solutions
- Consider scenarios
- If necessary, quantify
- Evaluate
- Evaluate subjectively
- Evaluate objectively
- Establish traceability

Name	
Description	
Type	
Topic	
Agent	
Viewpoint	
Priority	
Affected NFRs	
Affecting NFRs/Operationalizations	
Claim	
Sat Status	

Appendix

- RUP Specification
- Volere Specification
- How to Augment UML



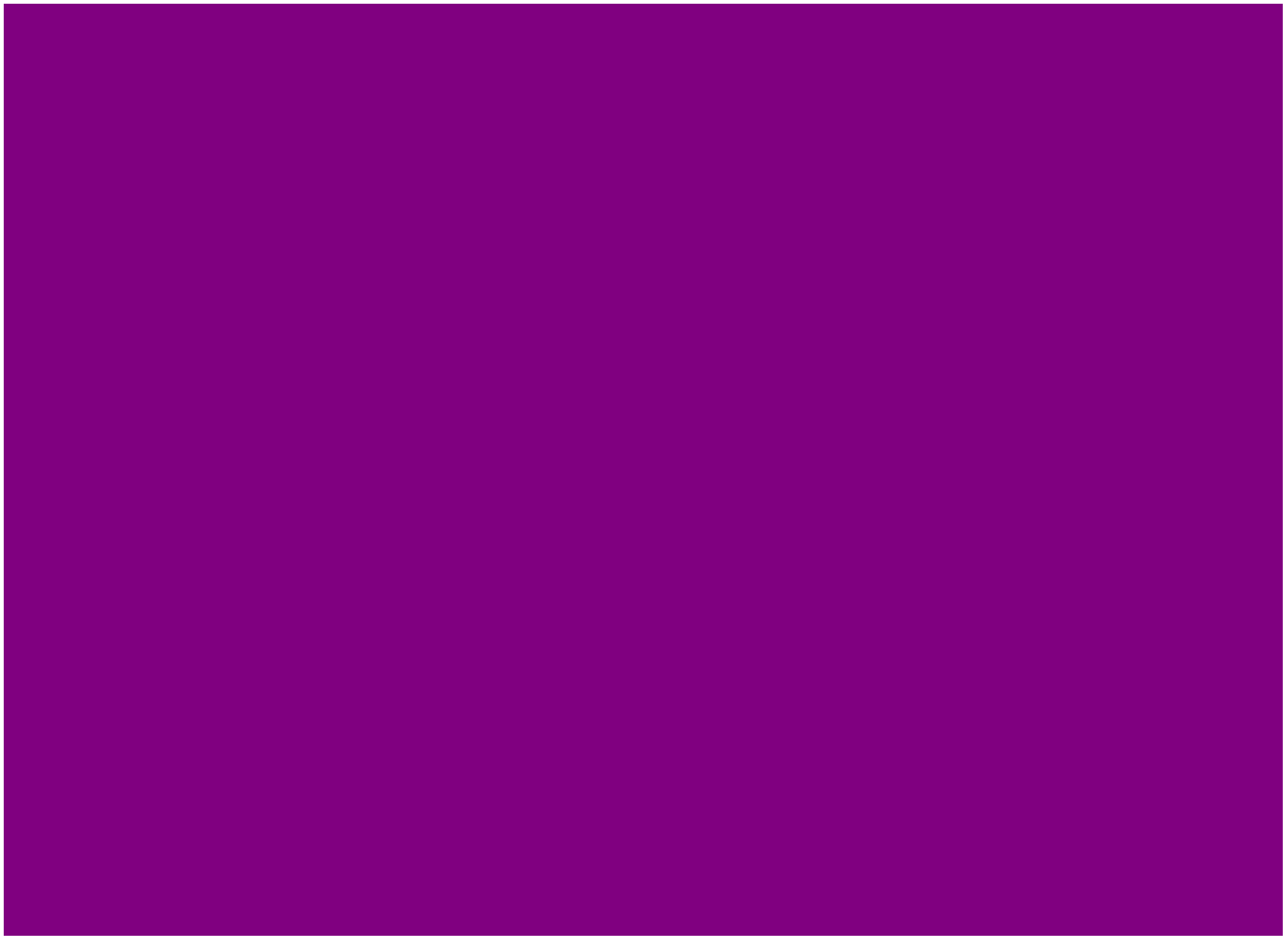
NFRs:
With Rational Unified Process and UML

Home Appliance Control System
Vision
Version 1.2

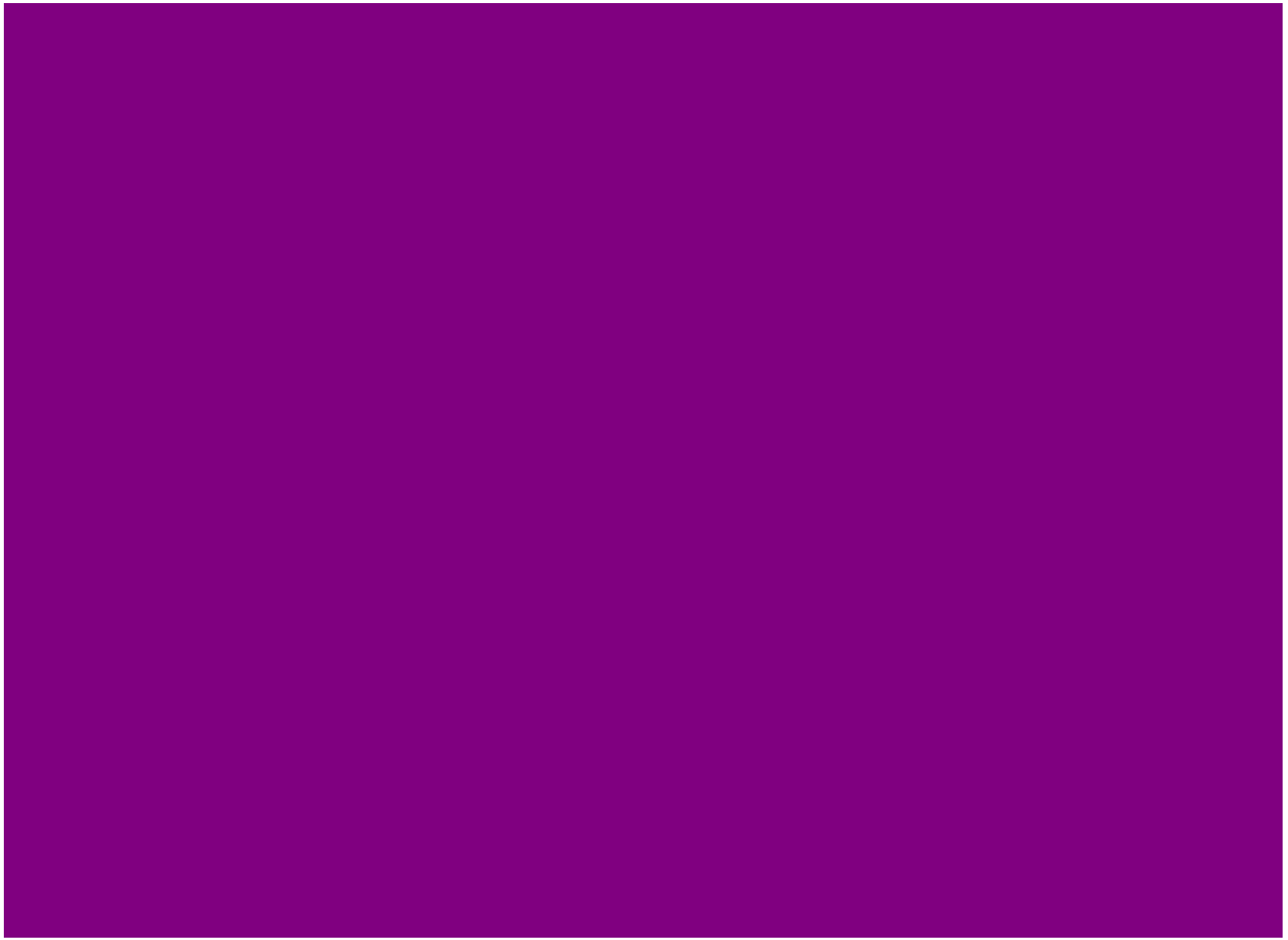
Revision History

Date	Version	Description	Author

NFRs:
With Rational Unified Process and UML

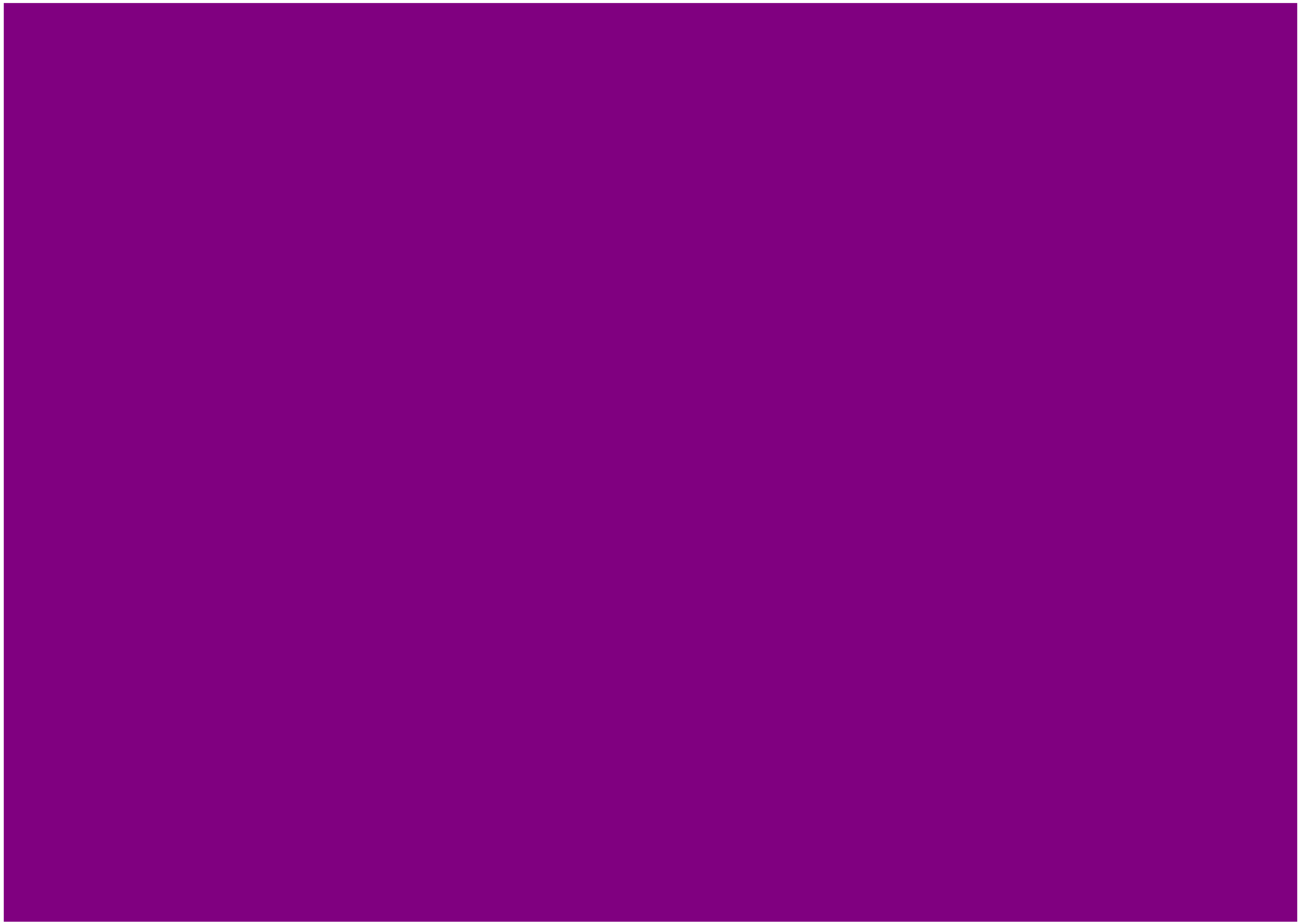


NFRs:
With Rational Unified Process and UML



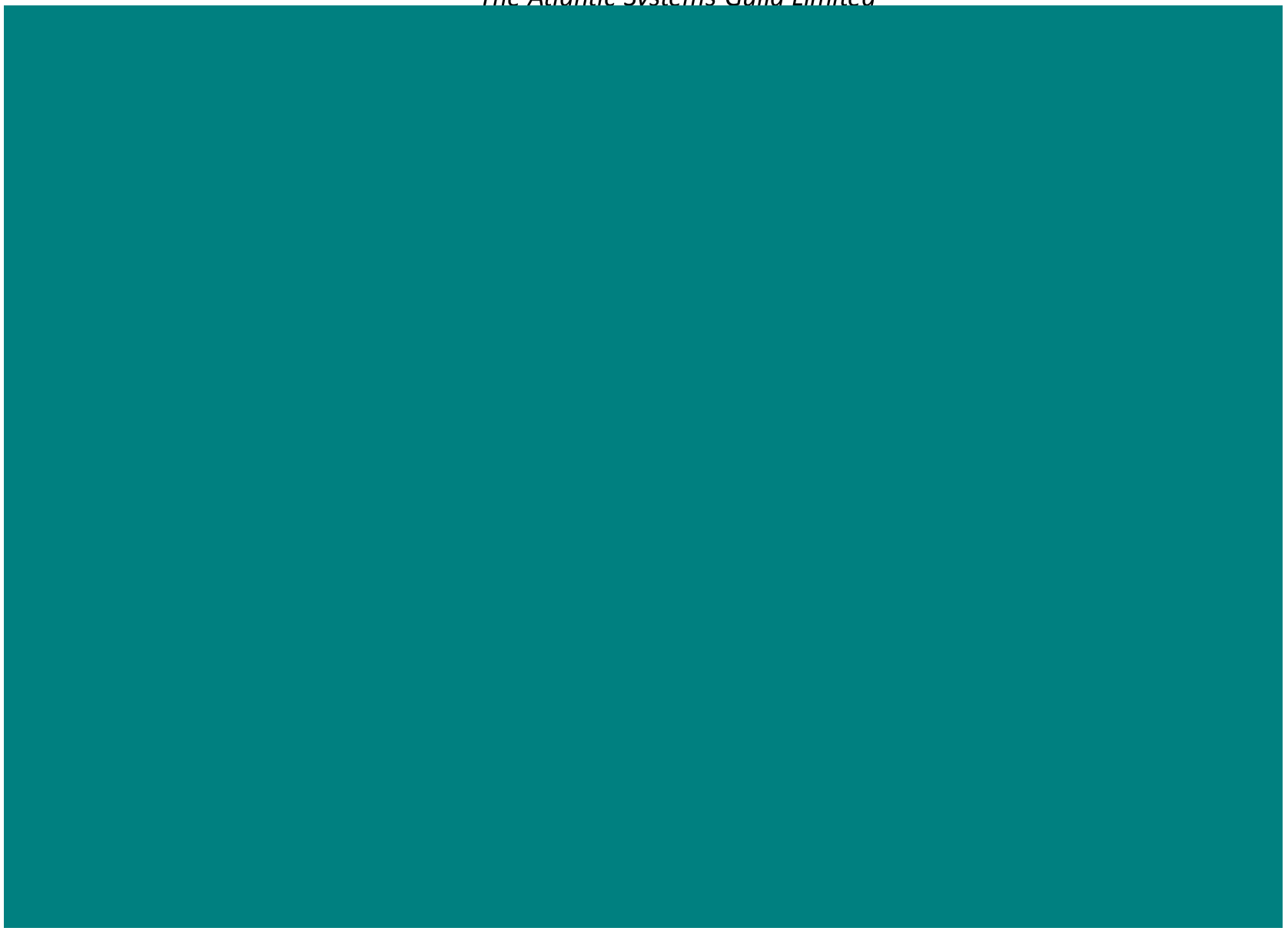
NFRs:
With Rational Unified Process and UML

NFRs:
With Rational Unified Process and UML



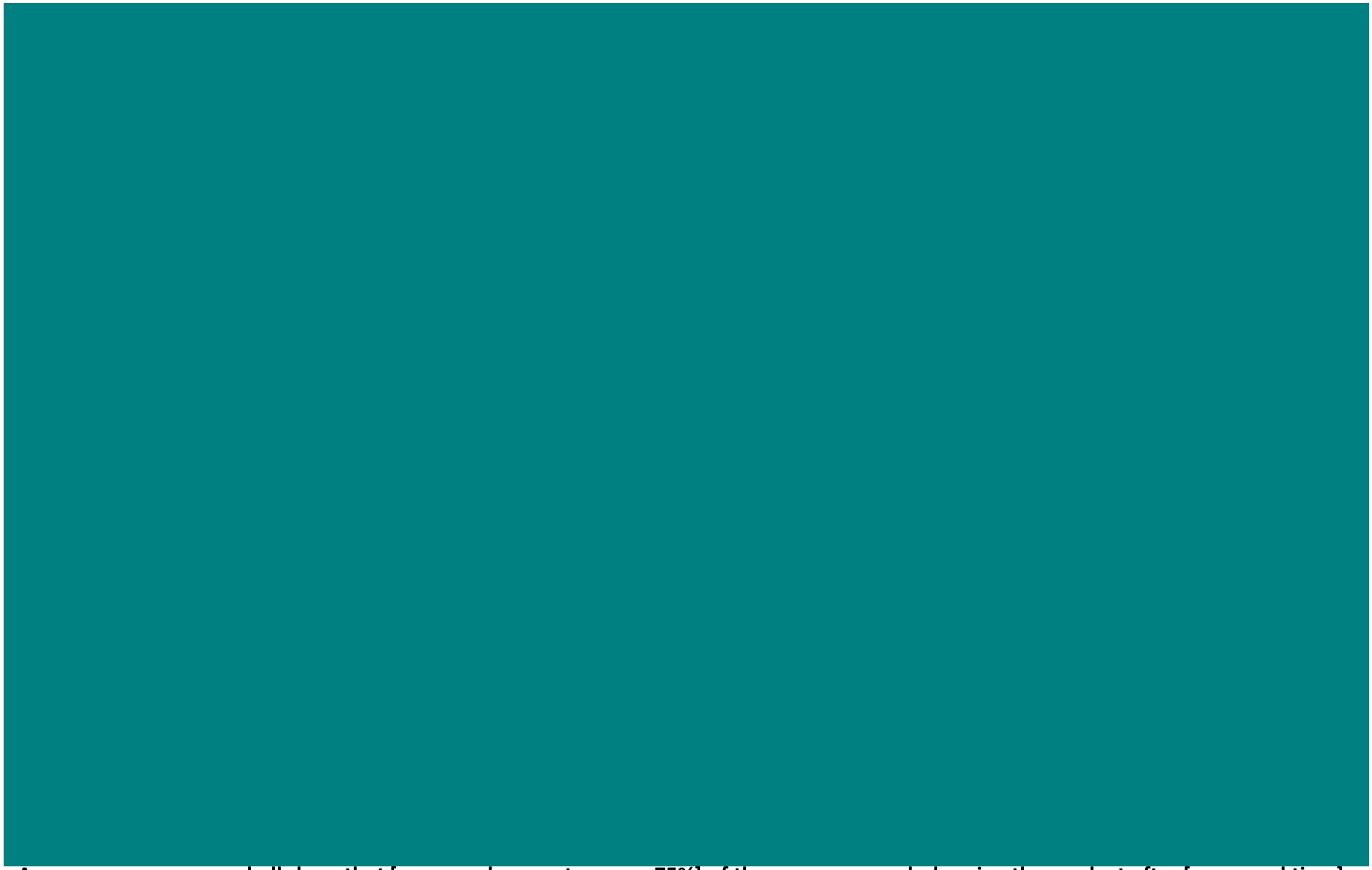
NFRs:
With Volere Requirements Specification Template

The Atlantic Systems Guild Limited



NFRs:
With Volere Requirements Specification Template

NFRs:
With Volere Requirements Specification Template



An anonymous survey shall show that [an agreed percentage, say 75%] of the users are regularly using the product after [an agreed time]

NFRs:
With Volere Requirements Specification Template

NFRs:
With Volere Requirements Specification Template

NFRs:
With Volere Requirements Specification Template

NFRs:
With Volere Requirements Specification Template

NFRs:
With Volere Requirements Specification Template

NFRs:
With Volere Requirements Specification Template

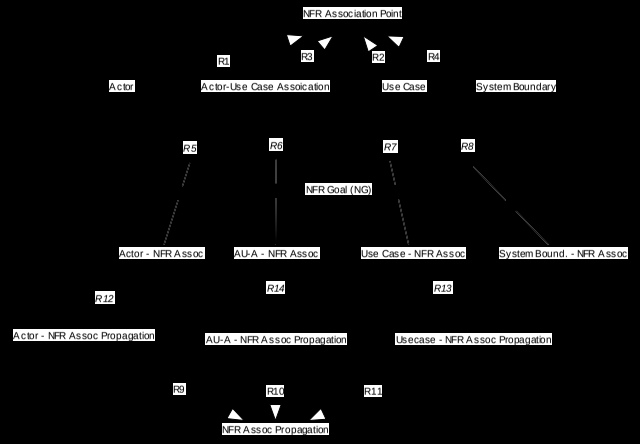
NFRs:
With Volere Requirements Specification Template

NFRs:
With Volere Requirements Specification Template

NFRs:
With Volere Requirements Specification Template

NFRs:
With Volere Requirements Specification Template

NFRs:
With Rational Unified Process and UML



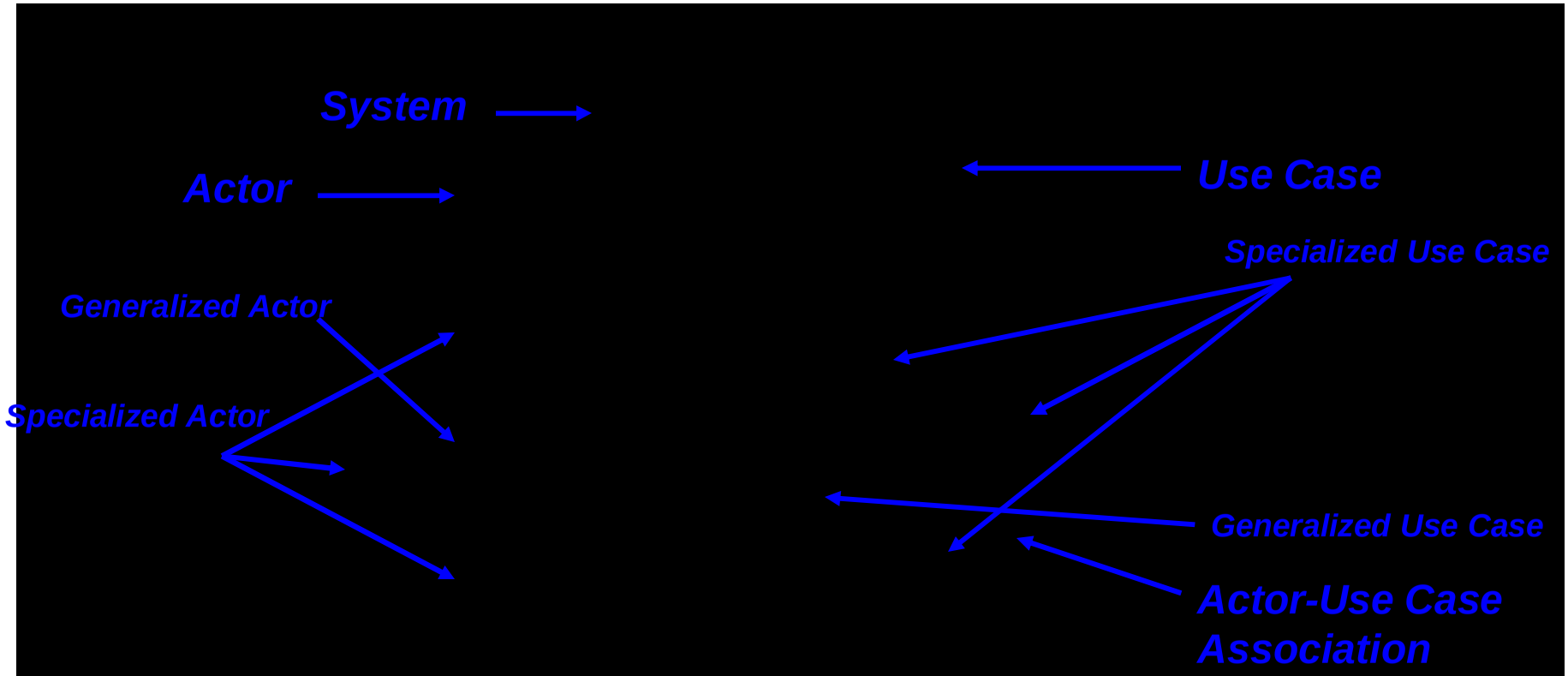
A Meta-model for partial FRs and NFRs Integration

Use cases as primary tool for FRs elicitation and modeling

Package Dependency Diagram **Class diagram** to describe components/objects and their relationships

Use cases are realized with **interaction diagram** showing interaction between components or objects

NFRs:
With Rational Unified Process and UML



System = the system in question that provides the functionality represented by use cases

Actor = an external entity (human or system)

Use case functionality (FRs) provided by the system

Actor-Use Case Association = an interface between an actor and the system

Use case details, including NFRs, are embedded textually using a template

NFRs:
With Rational Unified Process and UML

Inadequate Handling of NFRs

Textual description for NFRs embedded in the use case special requirements section – not 1st class citizens

Problems:

1. NFRs not modeled and organized, and not *visually*
2. NFRs not traceable to architecture and design
3. Error prone if NFR applicable to multiple use cases

NFRs:
With Rational Unified Process and UML

Other Integration Schemes

No single scheme providing all of:

- Use case driven
- Modeling constructs for representing and organizing NFRs
- Preserving underlying use case principles (e.g., ovals for FRs but not for NFRs)
- Generic for a wide range of NFRs